Problem_1a

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[3]: (436, 13)

[4]:
```
df_test.shape
```

[4]: (109, 13)

[5]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
df_oneAtrain = df_train[num_vars]
```

```
df_oneAtest = df_test[num_vars]
df_oneAtrain.head()
```

[5]:

	area	bedrooms	bathrooms	stories	parking	price
542	3620	2	1	1	0	1750000
496	4000	2	1	1	0	2695000
484	3040	2	1	1	0	2870000
507	3600	2	1	1	0	2590000
252	9860	3	1	1	0	4515000

[6]:
```
dataset_train = df_oneAtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[ 3620     2     1     1     0 1750000]
 [ 4000     2     1     1     0 2695000]
 [ 3040     2     1     1     0 2870000]
 [ 3600     2     1     1     0 2590000]
 [ 9860     3     1     1     0 4515000]
 [ 3968     3     1     2     0 4410000]
 [ 3840     3     1     2     1 4585000]
 [ 9800     4     2     2     2 5250000]
 [ 3640     2     1     1     0 3570000]
 [ 3520     2     2     1     0 3640000]
 [13200     3     1     2     2 9800000]
 [ 2700     2     1     1     0 2940000]
 [ 4300     6     2     2     0 6083000]
 [ 4500     2     1     1     0 3255000]
 [ 4995     4     2     1     0 4893000]
 [ 3069     2     1     1     1 3150000]
 [ 4352     4     1     2     1 2975000]
 [ 8880     3     2     2     1 6930000]
 [12944     3     1     1     0 3500000]
 [ 7160     3     1     1     2 5880000]]
```

[7]:
```
X_train = df_oneAtrain.values[:,0:5]
Y_train = df_oneAtrain.values[:,5]
len(X_train), len(Y_train)
```

[7]: (436, 436)

[8]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[3620    2    1    1    0]
 [4000    2    1    1    0]
 [3040    2    1    1    0]
 [3600    2    1    1    0]
 [9860    3    1    1    0]]
Y = [1750000 2695000 2870000 2590000 4515000]
```

[9]:
```
# Convert to 2D array (381x5)
m = len(X_train)
X_1 = X_train.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[3620    2    1    1    0]
 [4000    2    1    1    0]
 [3040    2    1    1    0]
 [3600    2    1    1    0]
 [9860    3    1    1    0]]
```

[10]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[10]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[11]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[11]:
```
array([[1.00e+00, 3.62e+03, 2.00e+00, 1.00e+00, 1.00e+00, 0.00e+00],
       [1.00e+00, 4.00e+03, 2.00e+00, 1.00e+00, 1.00e+00, 0.00e+00],
       [1.00e+00, 3.04e+03, 2.00e+00, 1.00e+00, 1.00e+00, 0.00e+00],
       [1.00e+00, 3.60e+03, 2.00e+00, 1.00e+00, 1.00e+00, 0.00e+00],
       [1.00e+00, 9.86e+03, 3.00e+00, 1.00e+00, 1.00e+00, 0.00e+00]])
```

[12]:
```
theta = np.zeros((6,1))
theta
```

[12]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[13]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
```

```
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[14]:
```
cost = compute_loss(X_train, Y_train, theta)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 5770455632864301.0
```

[15]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[16]:
```
theta = [0., 0., 0., 0., 0., 0.]
iterations = 2000
alpha = 0.0000000001
```

[17]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [2.10706875e-01 8.59342478e+02 6.95613564e-01
3.37363756e-01
 4.94874817e-01 1.85598518e-01]
loss_history = [1.31633712e+13 1.30921973e+13 1.30214655e+13 … 1.70465861e+12
 1.70465832e+12 1.70465804e+12]
```

[18]:
```
df_oneAtest.head()
```

[18]:

	area	bedrooms	bathrooms	stories	parking	price
239	4000	3	1	2	1	4585000
113	9620	3	1	1	2	6083000
325	3460	4	1	2	0	4007500
66	13200	2	1	1	1	6930000
479	3660	4	1	2	0	2940000

[19]:
```
dataset_test = df_oneAtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[  4000      3      1      2      1 4585000]
 [  9620      3      1      1      2 6083000]
 [  3460      4      1      2      0 4007500]
 [ 13200      2      1      1      1 6930000]
 [  3660      4      1      2      0 2940000]
 [  6350      3      2      3      0 6195000]
 [  3850      3      1      1      2 3535000]
 [  3480      3      1      2      1 2940000]
 [  3512      2      1      1      1 3500000]
 [  9000      4      2      4      2 7980000]
 [  6000      4      2      4      0 6755000]
 [  3960      3      1      2      0 3990000]
 [  3450      3      1      2      0 3150000]
 [  6060      3      1      1      0 3290000]
 [  5985      3      1      1      0 4130000]
 [  2430      3      1      1      0 2660000]
 [  4900      2      1      2      0 4410000]
 [  6020      3      1      1      0 3710000]
 [  3100      3      1      2      0 3360000]
 [  4500      2      1      1      2 4270000]]
```

[20]:
```
X_test = df_oneAtest.values[:,0:5]
Y_test = df_oneAtest.values[:,5]
len(X_test), len(Y_test)
```

[20]:
```
(109, 109)
```

[21]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[ 4000     3     1     2     1]
 [ 9620     3     1     1     2]
 [ 3460     4     1     2     0]
 [13200     2     1     1     1]
 [ 3660     4     1     2     0]]
Y = [4585000 6083000 4007500 6930000 2940000]
```

[22]:
```
# Convert to 2D array (164x5)
m = len(X_test)
X_1 = X_test.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[ 4000     3     1     2     1]
 [ 9620     3     1     1     2]
 [ 3460     4     1     2     0]
 [13200     2     1     1     1]
 [ 3660     4     1     2     0]]
```

[23]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[23]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[24]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[24]:
```
array([[1.00e+00, 4.00e+03, 3.00e+00, 1.00e+00, 2.00e+00, 1.00e+00],
       [1.00e+00, 9.62e+03, 3.00e+00, 1.00e+00, 1.00e+00, 2.00e+00],
       [1.00e+00, 3.46e+03, 4.00e+00, 1.00e+00, 2.00e+00, 0.00e+00],
       [1.00e+00, 1.32e+04, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00],
       [1.00e+00, 3.66e+03, 4.00e+00, 1.00e+00, 2.00e+00, 0.00e+00]])
```

[25]:
```
theta_test = np.zeros((6,1))
theta_test
```

[25]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[26]:
```
cost_test = compute_loss(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 1372813785875000.2
```

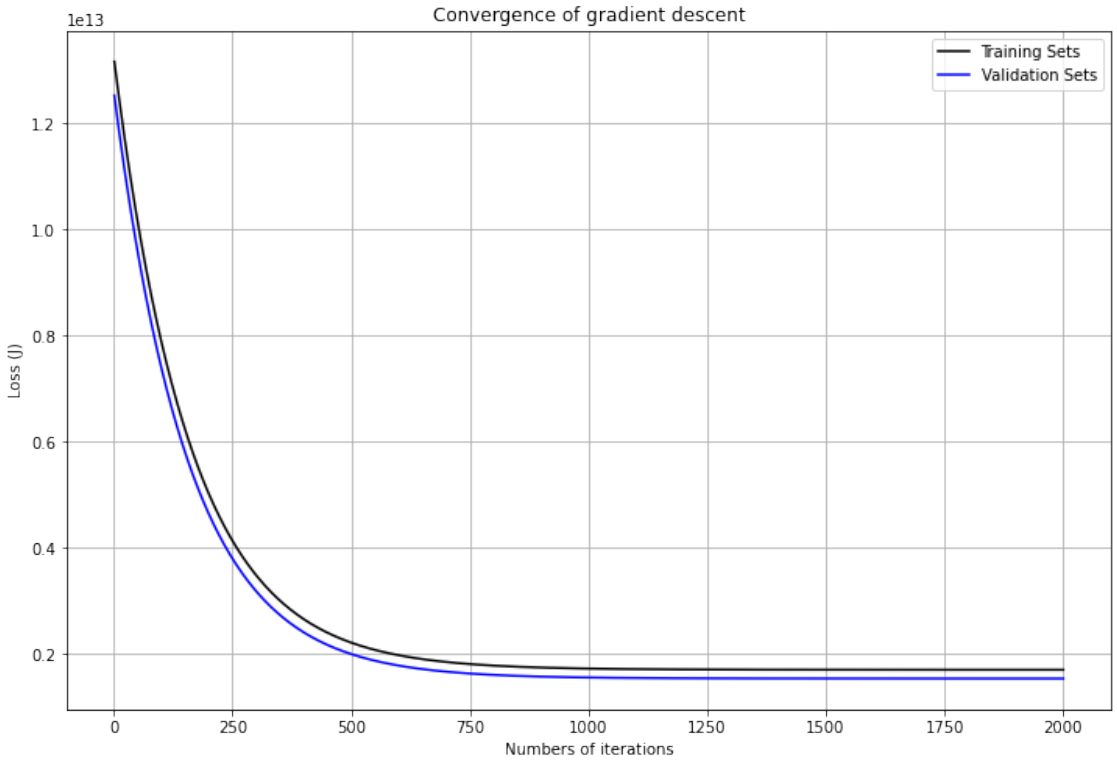
[27]:
```
theta_test = [0., 0., 0., 0., 0., 0.]
iterations = 2000
alpha = 0.0000000001
```

[28]:
```
theta_test, loss_history_test = gradient_descent(X_test, Y_test, theta_test,␣
 ↪alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [2.20563091e-01 8.33364199e+02 7.01107689e-01
3.53112759e-01
 5.07468404e-01 1.38571784e-01]
loss_history = [1.25245899e+13 1.24550016e+13 1.23858541e+13 … 1.53978999e+12
 1.53978978e+12 1.53978956e+12]
```

[30]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[30]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```


Convergence of gradient descent
1e13
Training Sets
Validation Sets
Loss (J)
Numbers of iterations
0
250
500
750
1000
1250
1500
1750
2000
0.2
0.4
0.6
0.8
1.0
1.2

[ ]:

Problem_1b

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Any dataset that has columns with values as 'Yes' or 'No', strings' values␣
 ↪cannot be used.
# However, we can convert them to numerical values as binary.

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished

```
2  12250000  9960         3          2        2         1          0
3  12215000  7500         4          2        2         1          0
4  11410000  7420         4          1        2         1          1

   basement  hotwaterheating  airconditioning  parking  prefarea  \
0         0                0                1        2         1
1         0                0                1        3         0
2         1                0                0        2         1
3         1                0                1        3         1
4         1                0                1        2         0

  furnishingstatus
0        furnished
1        furnished
2   semi-furnished
3        furnished
4        furnished
```

[4]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[4]:
```
(436, 13)
```

[5]:
```
df_test.shape
```

[5]:
```
(109, 13)
```

[6]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',␣
 ↪'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',␣
 ↪'prefarea', 'price']
df_oneBtrain = df_train[num_vars]
df_oneBtest = df_test[num_vars]
df_oneBtrain.head()
```

[6]:
```
     area  bedrooms  bathrooms  stories  mainroad  guestroom  basement  \
542  3620         2          1        1         1          0         0
496  4000         2          1        1         1          0         0
484  3040         2          1        1         0          0         0
507  3600         2          1        1         1          0         0
252  9860         3          1        1         1          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
```

```
542                0                0        0         0  1750000
496                0                0        0         0  2695000
484                0                0        0         0  2870000
507                0                0        0         0  2590000
252                0                0        0         0  4515000
```

[7]:
```
dataset_train = df_oneBtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[   3620       2       1       1       1       0       0       0       0
        0       0 1750000]
 [   4000       2       1       1       1       0       0       0       0
        0       0 2695000]
 [   3040       2       1       1       0       0       0       0       0
        0       0 2870000]
 [   3600       2       1       1       1       0       0       0       0
        0       0 2590000]
 [   9860       3       1       1       1       0       0       0       0
        0       0 4515000]
 [   3968       3       1       2       0       0       0       0       0
        0       0 4410000]
 [   3840       3       1       2       1       0       0       0       0
        1       1 4585000]
 [   9800       4       2       2       1       1       0       0       0
        2       0 5250000]
 [   3640       2       1       1       1       0       0       0       0
        0       0 3570000]
 [   3520       2       2       1       1       0       1       0       0
        0       0 3640000]
 [  13200       3       1       2       1       0       1       0       1
        2       1 9800000]
 [   2700       2       1       1       0       0       0       0       0
        0       0 2940000]
 [   4300       6       2       2       1       0       0       0       0
        0       0 6083000]
 [   4500       2       1       1       0       0       0       0       0
        0       0 3255000]
 [   4995       4       2       1       1       0       1       0       0
        0       0 4893000]
 [   3069       2       1       1       1       0       0       0       0
        1       0 3150000]
 [   4352       4       1       2       0       0       0       0       0
        1       0 2975000]
 [   8880       3       2       2       1       0       1       0       1
        1       0 6930000]
 [  12944       3       1       1       1       0       0       0       0
        0       0 3500000]
 [   7160       3       1       1       1       0       1       0       0
```

```
        2       1 5880000]]
```

[8]:
```
X_train = df_oneBtrain.values[:,0:11]
Y_train = df_oneBtrain.values[:,11]
len(X_train), len(Y_train)
```

[8]:
```
(436, 436)
```

[9]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[3620    2    1    1    1    0    0    0    0    0    0]
 [4000    2    1    1    1    0    0    0    0    0    0]
 [3040    2    1    1    0    0    0    0    0    0    0]
 [3600    2    1    1    1    0    0    0    0    0    0]
 [9860    3    1    1    1    0    0    0    0    0    0]]
Y = [1750000 2695000 2870000 2590000 4515000]
```

[10]:
```
# Convert to 2D array (381x11)
m = len(X_train)
X_1 = X_train.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[3620    2    1    1    1    0    0    0    0    0    0]
 [4000    2    1    1    1    0    0    0    0    0    0]
 [3040    2    1    1    0    0    0    0    0    0    0]
 [3600    2    1    1    1    0    0    0    0    0    0]
 [9860    3    1    1    1    0    0    0    0    0    0]]
```

[11]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[11]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[12]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[12]:
```
array([[1.00e+00, 3.62e+03, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
        0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00],
       [1.00e+00, 4.00e+03, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
        0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00],
       [1.00e+00, 3.04e+03, 2.00e+00, 1.00e+00, 1.00e+00, 0.00e+00,
        0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00],
```

```
        [1.00e+00, 3.60e+03, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
         0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00],
        [1.00e+00, 9.86e+03, 3.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
         0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00]])
```

[13]:
```
theta = np.zeros((12,1))
theta
```

[13]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[14]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[15]:
```
cost = compute_loss(X_train, Y_train, theta)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 5770455632864301.0
```

[16]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[17]:
```
theta = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 2000
alpha = 0.0000000001
```

[18]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [2.10706865e-01 8.59342434e+02 6.95613532e-01
3.37363742e-01
 4.94874797e-01 1.90578920e-01 6.05768881e-02 1.00766168e-01
 1.92179725e-02 1.13756069e-01 1.85598510e-01 7.46453669e-02]
loss_history = [1.31633712e+13 1.30921973e+13 1.30214655e+13 … 1.70465846e+12
 1.70465818e+12 1.70465790e+12]
```

[19]:
```
df_oneBtest.head()
```

[19]:
```
      area  bedrooms  bathrooms  stories  mainroad  guestroom  basement  \
239   4000         3          1        2         1          0         0
113   9620         3          1        1         1          0         1
325   3460         4          1        2         1          0         0
66   13200         2          1        1         1          0         1
479   3660         4          1        2         0          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
239                0                0        1         0  4585000
113                0                0        2         1  6083000
325                0                1        0         0  4007500
66                 1                0        1         0  6930000
479                0                0        0         0  2940000
```

[20]:
```
dataset_test = df_oneBtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[  4000       3       1       2       1       0       0       0       0
       1       0 4585000]
 [  9620       3       1       1       1       0       1       0       0
       2       1 6083000]
 [  3460       4       1       2       1       0       0       0       1
       0       0 4007500]
 [ 13200       2       1       1       1       0       1       1       0
       1       0 6930000]
 [  3660       4       1       2       0       0       0       0       0
       0       0 2940000]
 [  6350       3       2       3       1       1       0       0       1
       0       0 6195000]
 [  3850       3       1       1       1       0       0       0       0
       2       0 3535000]
 [  3480       3       1       2       0       0       0       0       0
       1       0 2940000]
 [  3512       2       1       1       1       0       0       0       0
       1       1 3500000]
 [  9000       4       2       4       1       0       0       0       1
       2       0 7980000]
 [  6000       4       2       4       1       0       0       0       1
       0       0 6755000]
 [  3960       3       1       2       1       0       0       0       0
       0       0 3990000]
 [  3450       3       1       2       1       0       1       0       0
       0       0 3150000]
 [  6060       3       1       1       1       1       1       0       0
       0       0 3290000]
 [  5985       3       1       1       1       0       1       0       0
       0       0 4130000]
 [  2430       3       1       1       0       0       0       0       0
```

```
        0       0 2660000]
 [   4900       2       1       2       1       0       1       0       0
        0       0 4410000]
 [   6020       3       1       1       1       0       0       0       0
        0       0 3710000]
 [   3100       3       1       2       0       0       1       0       0
        0       0 3360000]
 [   4500       2       1       1       1       0       0       0       1
        2       0 4270000]]
```

[21]:
```
X_test = df_oneBtest.values[:,0:11]
Y_test = df_oneBtest.values[:,11]
len(X_test), len(Y_test)
```

[21]:
```
(109, 109)
```

[22]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[ 4000     3     1     2     1     0     0     0     0     1     0]
 [ 9620     3     1     1     1     0     1     0     0     2     1]
 [ 3460     4     1     2     1     0     0     0     1     0     0]
 [13200     2     1     1     1     0     1     1     0     1     0]
 [ 3660     4     1     2     0     0     0     0     0     0     0]]
Y = [4585000 6083000 4007500 6930000 2940000]
```

[23]:
```
# Convert to 2D array (164x11)
m = len(X_test)
X_1 = X_test.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[ 4000     3     1     2     1     0     0     0     0     1     0]
 [ 9620     3     1     1     1     0     1     0     0     2     1]
 [ 3460     4     1     2     1     0     0     0     1     0     0]
 [13200     2     1     1     1     0     1     1     0     1     0]
 [ 3660     4     1     2     0     0     0     0     0     0     0]]
```

[24]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[24]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[25]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[25]:
```
array([[1.00e+00, 4.00e+03, 3.00e+00, 1.00e+00, 2.00e+00, 1.00e+00,
        0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 1.00e+00, 0.00e+00],
       [1.00e+00, 9.62e+03, 3.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
        0.00e+00, 1.00e+00, 0.00e+00, 0.00e+00, 2.00e+00, 1.00e+00],
       [1.00e+00, 3.46e+03, 4.00e+00, 1.00e+00, 2.00e+00, 1.00e+00,
        0.00e+00, 0.00e+00, 0.00e+00, 1.00e+00, 0.00e+00, 0.00e+00],
       [1.00e+00, 1.32e+04, 2.00e+00, 1.00e+00, 1.00e+00, 1.00e+00,
        0.00e+00, 1.00e+00, 1.00e+00, 0.00e+00, 1.00e+00, 0.00e+00],
       [1.00e+00, 3.66e+03, 4.00e+00, 1.00e+00, 2.00e+00, 0.00e+00,
        0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00, 0.00e+00]])
```

[26]:
```
theta_test = np.zeros((12,1))
theta_test
```

[26]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[27]:
```
cost_test = compute_loss(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 1372813785875000.2
```

[28]:
```
theta_test = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 2000
alpha = 0.0000000001
```

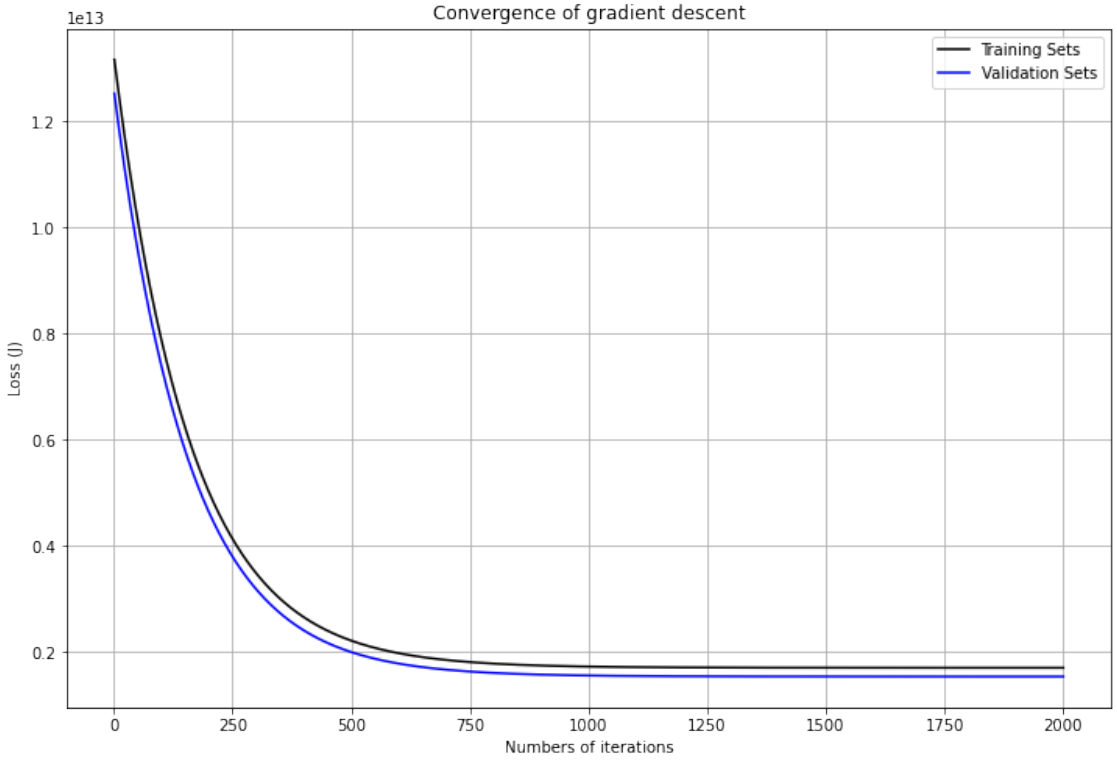
[29]:
```
theta_test, loss_history_test = gradient_descent(X_test, Y_test, theta_test,␣
 ↪alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [2.20563080e-01 8.33364158e+02 7.01107658e-01
3.53112746e-01
 5.07468385e-01 1.87064669e-01 4.59277714e-02 1.02690815e-01
 1.20041892e-02 1.12130581e-01 1.38571780e-01 5.56484402e-02]
```

```
loss_history = [1.25245899e+13 1.24550016e+13 1.23858540e+13 … 1.53978986e+12
 1.53978964e+12 1.53978943e+12]
```

[31]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[31]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]:

Problem_2a_MinMax

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[3]: (436, 13)

[4]:
```
df_test.shape
```

[4]: (109, 13)

[5]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
df_twoAtrain = df_train[num_vars]
```

```
df_twoAtest = df_test[num_vars]
df_twoAtrain.head()
```

[5]:

	area	bedrooms	bathrooms	stories	parking	price
542	3620	2	1	1	0	1750000
496	4000	2	1	1	0	2695000
484	3040	2	1	1	0	2870000
507	3600	2	1	1	0	2590000
252	9860	3	1	1	0	4515000

[6]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_twoAtrain[num_vars] = scaler.fit_transform(df_twoAtrain[num_vars])
df_twoAtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	parking	price
542	0.124199	0.2	0.0	0.0	0.0	0.000000
496	0.150654	0.2	0.0	0.0	0.0	0.081818
484	0.083821	0.2	0.0	0.0	0.0	0.096970
507	0.122807	0.2	0.0	0.0	0.0	0.072727
252	0.558619	0.4	0.0	0.0	0.0	0.239394

[7]:
```
dataset_train = df_twoAtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[0.12419939 0.2        0.         0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         0.         0.08181818]
 [0.08382066 0.2        0.         0.         0.         0.0969697 ]
 [0.12280702 0.2        0.         0.         0.         0.07272727]
 [0.55861877 0.4        0.         0.         0.         0.23939394]
 [0.14842662 0.4        0.         0.33333333 0.         0.23030303]
 [0.13951546 0.4        0.         0.33333333 0.33333333 0.24545455]
 [0.55444166 0.6        0.5        0.33333333 0.66666667 0.3030303 ]
 [0.12559176 0.2        0.         0.         0.         0.15757576]
 [0.11723754 0.2        0.5        0.         0.         0.16363636]
 [0.79114453 0.4        0.         0.33333333 0.66666667 0.6969697 ]
```

```
 [0.06015038 0.2        0.         0.         0.         0.1030303 ]
 [0.17153996 1.         0.5        0.33333333 0.         0.37515152]
 [0.18546366 0.2        0.         0.         0.         0.13030303]
 [0.21992481 0.6        0.5        0.         0.         0.27212121]
 [0.0858396  0.2        0.         0.         0.33333333 0.12121212]
 [0.17516012 0.6        0.         0.33333333 0.33333333 0.10606061]
 [0.49039265 0.4        0.5        0.33333333 0.33333333 0.44848485]
 [0.77332219 0.4        0.         0.         0.         0.15151515]
 [0.37064884 0.4        0.         0.         0.66666667 0.35757576]]
```

[8]:
```
X_train = df_twoAtrain.values[:,0:5]
Y_train = df_twoAtrain.values[:,5]
len(X_train), len(Y_train)
```

[8]:
```
(436, 436)
```

[9]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[0.12419939 0.2        0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         0.        ]
 [0.08382066 0.2        0.         0.         0.        ]
 [0.12280702 0.2        0.         0.         0.        ]
 [0.55861877 0.4        0.         0.         0.        ]]
Y = [0.         0.08181818 0.0969697  0.07272727 0.23939394]
```

[10]:
```
# Convert to 2D array (381x5)
m = len(X_train)
X_1 = X_train.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.12419939 0.2        0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         0.        ]
 [0.08382066 0.2        0.         0.         0.        ]
 [0.12280702 0.2        0.         0.         0.        ]
 [0.55861877 0.4        0.         0.         0.        ]]
```

[11]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[11]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[12]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[12]:
```
array([[1.        , 0.12419939, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.15065441, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.08382066, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.12280702, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.55861877, 0.4       , 0.        , 0.        ,
        0.        ]])
```

[13]:
```
theta = np.zeros((6,1))
theta
```

[13]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[14]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[15]:
```
cost = compute_loss(X_train, Y_train, theta)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 20.934727519081726
```

[16]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[17]:
```
theta = [0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[18]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [0.15859029 0.08182965 0.08651773 0.0835067  0.09310226
0.08501595]
loss_history = [0.04770179 0.04739073 0.04708224 … 0.0085317  0.00853091
0.00853011]
```

[19]:
```
df_twoAtest.head()
```

[19]:

	area	bedrooms	bathrooms	stories	parking	price
239	4000	3	1	2	1	4585000
113	9620	3	1	1	2	6083000
325	3460	4	1	2	0	4007500
66	13200	2	1	1	1	6930000
479	3660	4	1	2	0	2940000

[20]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_twoAtest[num_vars] = scaler.fit_transform(df_twoAtest[num_vars])
df_twoAtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	parking	price
239	0.203463	0.50	0.0	0.333333	0.333333	0.270000
113	0.690043	0.50	0.0	0.000000	0.666667	0.412667
325	0.156710	0.75	0.0	0.333333	0.000000	0.215000
66	1.000000	0.25	0.0	0.000000	0.333333	0.493333
479	0.174026	0.75	0.0	0.333333	0.000000	0.113333

[21]:
```
dataset_test = df_twoAtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[0.2034632  0.5        0.         0.33333333 0.33333333 0.27      ]
 [0.69004329 0.5        0.         0.         0.66666667 0.41266667]
 [0.15670996 0.75       0.         0.33333333 0.         0.215     ]
 [1.         0.25       0.         0.         0.33333333 0.49333333]
 [0.17402597 0.75       0.         0.33333333 0.         0.11333333]
 [0.40692641 0.5        0.33333333 0.66666667 0.         0.42333333]
 [0.19047619 0.5        0.         0.         0.66666667 0.17      ]
 [0.15844156 0.5        0.         0.33333333 0.33333333 0.11333333]
 [0.16121212 0.25       0.         0.         0.33333333 0.16666667]
 [0.63636364 0.75       0.33333333 1.         0.66666667 0.59333333]
 [0.37662338 0.75       0.33333333 1.         0.         0.47666667]
 [0.2        0.5        0.         0.33333333 0.         0.21333333]
 [0.15584416 0.5        0.         0.33333333 0.         0.13333333]
 [0.38181818 0.5        0.         0.         0.         0.14666667]
```

```
 [0.37532468 0.5        0.         0.         0.         0.22666667]
 [0.06753247 0.5        0.         0.         0.         0.08666667]
 [0.28138528 0.25       0.         0.33333333 0.         0.25333333]
 [0.37835498 0.5        0.         0.         0.         0.18666667]
 [0.12554113 0.5        0.         0.33333333 0.         0.15333333]
 [0.24675325 0.25       0.         0.         0.66666667 0.24      ]]
```

[22]:
```
X_test = df_twoAtest.values[:,0:5]
Y_test = df_twoAtest.values[:,5]
len(X_test), len(Y_test)
```

[22]:
```
(109, 109)
```

[23]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[0.2034632  0.5        0.         0.33333333 0.33333333]
 [0.69004329 0.5        0.         0.         0.66666667]
 [0.15670996 0.75       0.         0.33333333 0.        ]
 [1.         0.25       0.         0.         0.33333333]
 [0.17402597 0.75       0.         0.33333333 0.        ]]
Y = [0.27       0.41266667 0.215      0.49333333 0.11333333]
```

[24]:
```
# Convert to 2D array (164x5)
m = len(X_test)
X_1 = X_test.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.2034632  0.5        0.         0.33333333 0.33333333]
 [0.69004329 0.5        0.         0.         0.66666667]
 [0.15670996 0.75       0.         0.33333333 0.        ]
 [1.         0.25       0.         0.         0.33333333]
 [0.17402597 0.75       0.         0.33333333 0.        ]]
```

[25]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[25]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[26]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[26]:
```
array([[1.        , 0.2034632 , 0.5       , 0.        , 0.33333333,
        0.33333333],
       [1.        , 0.69004329, 0.5       , 0.        , 0.        ,
        0.66666667],
       [1.        , 0.15670996, 0.75      , 0.        , 0.33333333,
        0.        ],
       [1.        , 1.        , 0.25      , 0.        , 0.        ,
        0.33333333],
       [1.        , 0.17402597, 0.75      , 0.        , 0.33333333,
        0.        ]])
```

[27]:
```
theta_test = np.zeros((6,1))
theta_test
```

[27]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[28]:
```
cost_test = compute_loss(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 5.79399238888889
```

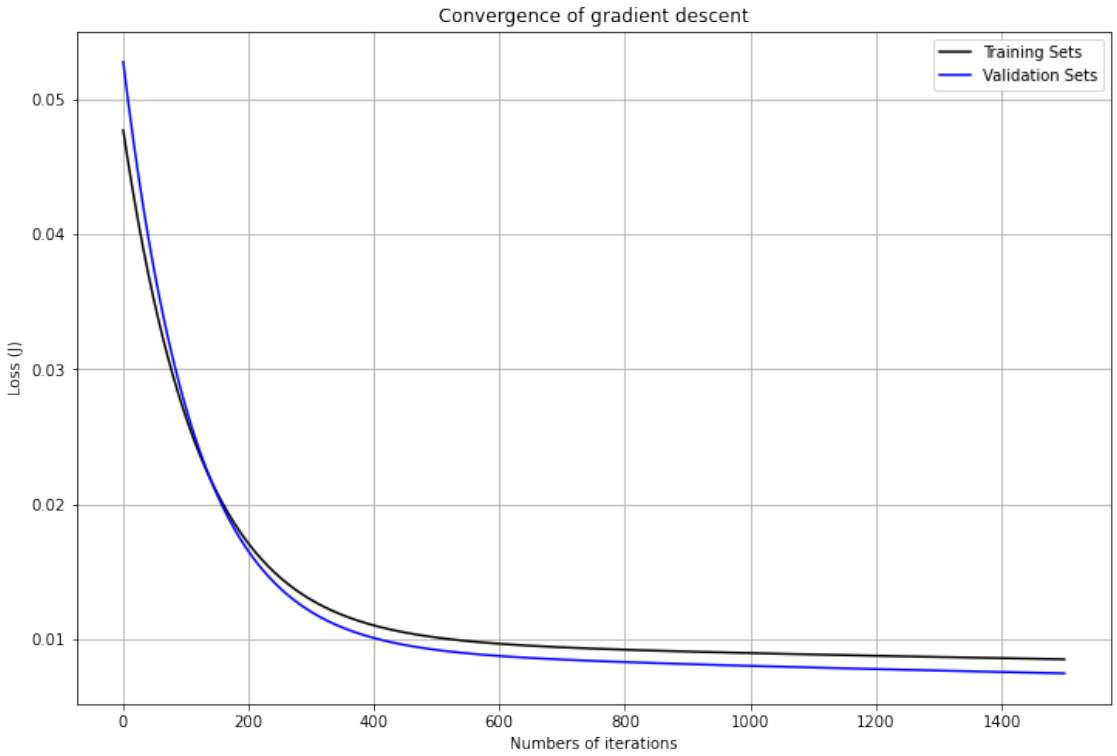
[29]:
```
theta_test = [0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[30]:
```
theta_test, loss_history_test = gradient_descent(X_test, Y_test, theta_test,␣
 ↪alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [0.161425   0.10594748 0.09248631 0.07041321 0.09856776
0.08675523]
loss_history = [0.05276804 0.0523836  0.05200255 … 0.00748962 0.00748864
0.00748766]
```

[32]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[32]: Text(0.5, 1.0, 'Convergence of gradient descent')



[ ]:

Problem_2a_Standardization

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[3]: (436, 13)

[4]:
```
df_test.shape
```

[4]: (109, 13)

[5]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
df_twoAtrain = df_train[num_vars]
```

```
df_twoAtest = df_test[num_vars]
df_twoAtrain.head()
```

[5]:

	area	bedrooms	bathrooms	stories	parking	price
542	3620	2	1	1	0	1750000
496	4000	2	1	1	0	2695000
484	3040	2	1	1	0	2870000
507	3600	2	1	1	0	2590000
252	9860	3	1	1	0	4515000

[6]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_twoAtrain[num_vars] = scaler.fit_transform(df_twoAtrain[num_vars])
df_twoAtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	parking	price
542	-0.716772	-1.294376	-0.573307	-0.933142	-0.819149	-1.586001
496	-0.538936	-1.294376	-0.573307	-0.933142	-0.819149	-1.090971
484	-0.988206	-1.294376	-0.573307	-0.933142	-0.819149	-0.999299
507	-0.726132	-1.294376	-0.573307	-0.933142	-0.819149	-1.145974
252	2.203478	0.052516	-0.573307	-0.933142	-0.819149	-0.137579

[7]:
```
dataset_train = df_twoAtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[-0.71677205 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -1.5860012 ]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -1.09097091]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.99929863]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -1.14597428]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164 -0.81914879 -0.13757923]
 [-0.55391195  0.05251643 -0.57330726  0.21291401 -0.81914879 -0.1925826 ]
 [-0.61381451  0.05251643 -0.57330726  0.21291401  0.32555914 -0.10091032]
 [ 2.17539862  1.39940847  1.4755613   0.21291401  1.47026706  0.24744433]
 [-0.70741227 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.63260953]
 [-0.76357092 -1.29437561  1.4755613  -0.93314164 -0.81914879 -0.59594061]
 [ 3.76656047  0.05251643 -0.57330726  0.21291401  1.47026706  2.63092353]
```

```
 [-1.14732172 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.96262972]
 [-0.39853968  4.09319255  1.4755613   0.21291401 -0.81914879  0.68380437]
 [-0.30494192 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.79761962]
 [-0.07328747  1.39940847  1.4755613  -0.93314164 -0.81914879  0.06043289]
 [-0.97463386 -1.29437561 -0.57330726 -0.93314164  0.32555914 -0.85262299]
 [-0.37420426  1.39940847 -0.57330726  0.21291401  0.32555914 -0.94429527]
 [ 1.74484894  0.05251643  1.4755613   0.21291401  0.32555914  1.12749819]
 [ 3.64675535  0.05251643 -0.57330726 -0.93314164 -0.81914879 -0.66927844]
 [ 0.93990824  0.05251643 -0.57330726 -0.93314164  1.47026706  0.57746453]]
```

[8]:
```
X_train = df_twoAtrain.values[:,0:5]
Y_train = df_twoAtrain.values[:,5]
len(X_train), len(Y_train)
```

[8]:
```
(436, 436)
```

[9]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164 -0.81914879]]
Y = [-1.5860012  -1.09097091 -0.99929863 -1.14597428 -0.13757923]
```

[10]:
```
# Convert to 2D array (381x5)
m = len(X_train)
X_1 = X_train.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164 -0.81914879]]
```

[11]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[11]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[12]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[12]:
```
array([[ 1.        , -0.71677205, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        , -0.53893631, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        , -0.98820554, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        , -0.72613182, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        ,  2.20347795,  0.05251643, -0.57330726, -0.93314164,
        -0.81914879]])
```

[13]:
```
theta = np.zeros((6,1))
theta
```

[13]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[14]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[15]:
```
cost = compute_loss(X_train, Y_train, theta)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 218.00000000000006
```

[16]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[17]:
```
theta = [0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[18]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [2.56815975e-16 3.81901939e-01 9.94453409e-02
2.99572284e-01
 2.34863362e-01 1.64796490e-01]
loss_history = [0.49701359 0.49406054 0.49114047 … 0.22316487 0.22316462
0.22316437]
```

[19]:
```
df_twoAtest.head()
```

[19]:

	area	bedrooms	bathrooms	stories	parking	price
239	4000	3	1	2	1	4585000
113	9620	3	1	1	2	6083000
325	3460	4	1	2	0	4007500
66	13200	2	1	1	1	6930000
479	3660	4	1	2	0	2940000

[20]:
```
# Many columns contains small integer values excluding areas. Needs to rescale the variables.
# Advised to use starndarization or normalization, so the coefficients is comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_twoAtest[num_vars] = scaler.fit_transform(df_twoAtest[num_vars])
df_twoAtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	parking	price
239	-0.500735	0.025607	-0.563545	0.272416	0.492144	-0.081358
113	1.954229	0.025607	-0.563545	-0.915317	1.739673	0.801114
325	-0.736621	1.421209	-0.563545	0.272416	-0.755384	-0.421563
66	3.518067	-1.369995	-0.563545	-0.915317	0.492144	1.300082
479	-0.649256	1.421209	-0.563545	0.272416	-0.755384	-1.050428

[21]:
```
dataset_test = df_twoAtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[-0.50073521  0.02560738 -0.56354451  0.27241586  0.49214421 -0.08135801]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  1.73967255  0.80111439]
 [-0.73662142  1.42120937 -0.56354451  0.27241586 -0.75538413 -0.42156349]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.49214421  1.30008243]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -0.75538413 -1.05042817]
 [ 0.52580664  0.02560738  1.2431129   1.46014902 -0.75538413  0.86709364]
 [-0.56625916  0.02560738 -0.56354451 -0.91531729  1.73967255 -0.69991343]
 [-0.72788489  0.02560738 -0.56354451  0.27241586  0.49214421 -1.05042817]
 [-0.71390645 -1.36999462 -0.56354451 -0.91531729  0.49214421 -0.72053194]
 [ 1.68339637  1.42120937  1.2431129   2.64788217  1.73967255  1.91863786]
 [ 0.37291742  1.42120937  1.2431129   2.64788217 -0.75538413  1.19698986]
 [-0.51820826  0.02560738 -0.56354451  0.27241586 -0.75538413 -0.43187275]
```

```
 [-0.74098968  0.02560738 -0.56354451  0.27241586 -0.75538413 -0.92671708]
 [ 0.399127    0.02560738 -0.56354451 -0.91531729 -0.75538413 -0.84424303]
 [ 0.36636503  0.02560738 -0.56354451 -0.91531729 -0.75538413 -0.34939869]
 [-1.18655253  0.02560738 -0.56354451 -0.91531729 -0.75538413 -1.21537628]
 [-0.10759152 -1.36999462 -0.56354451  0.27241586 -0.75538413 -0.18445058]
 [ 0.38165395  0.02560738 -0.56354451 -0.91531729 -0.75538413 -0.59682086]
 [-0.89387889  0.02560738 -0.56354451  0.27241586 -0.75538413 -0.803006  ]
 [-0.28232205 -1.36999462 -0.56354451 -0.91531729  1.73967255 -0.26692463]]
```

[22]:
```
X_test = df_twoAtest.values[:,0:5]
Y_test = df_twoAtest.values[:,5]
len(X_test), len(Y_test)
```

[22]:
```
(109, 109)
```

[23]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.49214421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  1.73967255]
 [-0.73662142  1.42120937 -0.56354451  0.27241586 -0.75538413]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.49214421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -0.75538413]]
Y = [-0.08135801  0.80111439 -0.42156349  1.30008243 -1.05042817]
```

[24]:
```
# Convert to 2D array (164x5)
m = len(X_test)
X_1 = X_test.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.49214421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  1.73967255]
 [-0.73662142  1.42120937 -0.56354451  0.27241586 -0.75538413]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.49214421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -0.75538413]]
```

[25]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[25]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[26]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[26]:
```
array([[ 1.        , -0.50073521,  0.02560738, -0.56354451,  0.27241586,
         0.49214421],
       [ 1.        ,  1.95422869,  0.02560738, -0.56354451, -0.91531729,
         1.73967255],
       [ 1.        , -0.73662142,  1.42120937, -0.56354451,  0.27241586,
        -0.75538413],
       [ 1.        ,  3.5180669 , -1.36999462, -0.56354451, -0.91531729,
         0.49214421],
       [ 1.        , -0.64925616,  1.42120937, -0.56354451,  0.27241586,
        -0.75538413]])
```

[27]:
```
theta_test = np.zeros((6,1))
theta_test
```

[27]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[28]:
```
cost_test = compute_loss(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 54.49999999999999
```

[29]:
```
theta_test = [0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

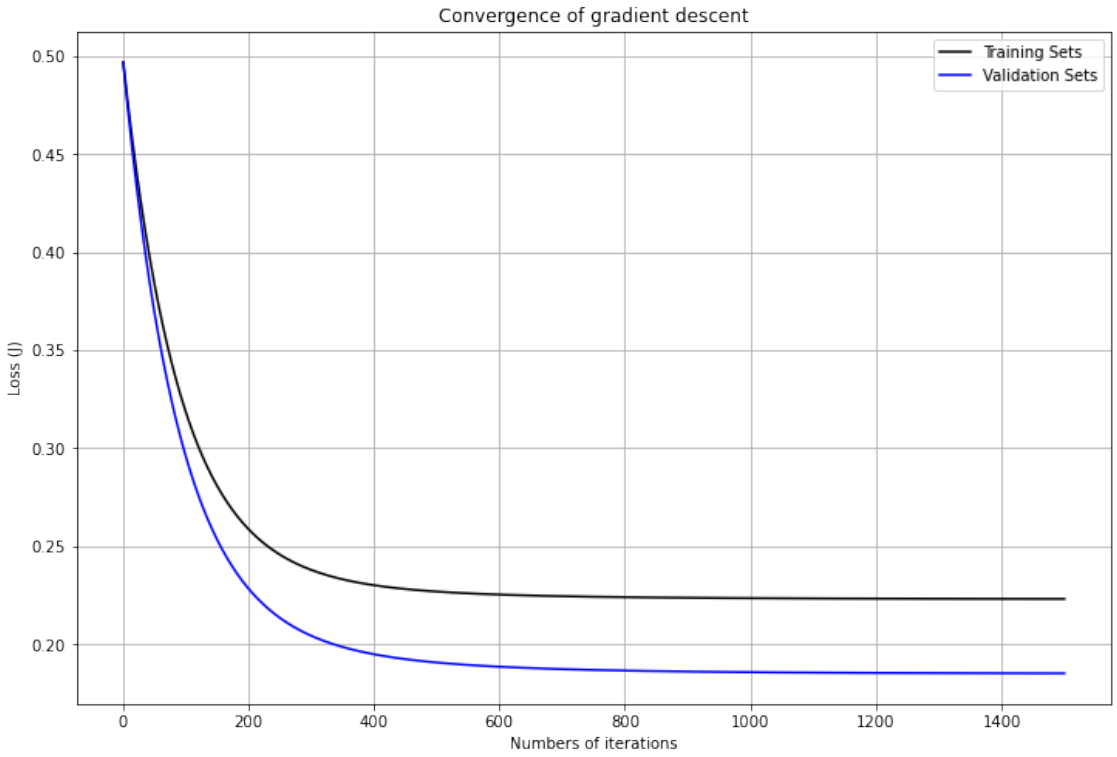
[30]:
```
theta_test, loss_history_test = gradient_descent(X_test, Y_test, theta_test,␣
 ↪alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [ 9.28431643e-17  3.58547244e-01 -3.04213335e-02
3.28020165e-01
  3.06752025e-01  2.34115221e-01]
loss_history = [0.49663315 0.49330422 0.49001277 … 0.18520046 0.18520007
0.18519967]
```

[32]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
```

```
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[32]: Text(0.5, 1.0, 'Convergence of gradient descent')



[ ]:

Problem_2b_MinMax

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Any dataset that has columns with values as 'Yes' or 'No', strings' values␣
 ↪cannot be used.
# However, we can convert them to numerical values as binary.

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished

[4]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[4]: (436, 13)

[5]:
```
df_test.shape
```

[5]: (109, 13)

[6]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',␣
 ↪'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',␣
 ↪'prefarea', 'price']
df_twoBtrain = df_train[num_vars]
df_twoBtest = df_test[num_vars]
df_twoBtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
542	3620	2	1	1	1	0	0	0	0	0	0	1750000
496	4000	2	1	1	1	0	0	0	0	0	0	2660000
484	3040	2	1	1	0	0	0	0	0	0	0	2800000
507	3600	2	1	1	1	0	0	0	0	0	0	2450000

```
252  9860         3          1        1         1          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
542                0                0        0         0  1750000
496                0                0        0         0  2695000
484                0                0        0         0  2870000
507                0                0        0         0  2590000
252                0                0        0         0  4515000
```

[7]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_twoBtrain[num_vars] = scaler.fit_transform(df_twoBtrain[num_vars])
df_twoBtrain.head()
```

[7]:
```
         area  bedrooms  bathrooms  stories  mainroad  guestroom  basement  \
542  0.124199       0.2        0.0      0.0       1.0        0.0       0.0
496  0.150654       0.2        0.0      0.0       1.0        0.0       0.0
484  0.083821       0.2        0.0      0.0       0.0        0.0       0.0
507  0.122807       0.2        0.0      0.0       1.0        0.0       0.0
252  0.558619       0.4        0.0      0.0       1.0        0.0       0.0

     hotwaterheating  airconditioning  parking  prefarea     price
542              0.0              0.0      0.0       0.0  0.000000
496              0.0              0.0      0.0       0.0  0.081818
484              0.0              0.0      0.0       0.0  0.096970
507              0.0              0.0      0.0       0.0  0.072727
252              0.0              0.0      0.0       0.0  0.239394
```

[8]:
```
dataset_train = df_twoBtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[0.12419939 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.         ]
 [0.15065441 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.08181818]
 [0.08382066 0.2        0.         0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.0969697 ]
[0.12280702 0.2        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.07272727]
[0.55861877 0.4        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.23939394]
[0.14842662 0.4        0.         0.33333333 0.         0.
 0.         0.         0.         0.         0.         0.23030303]
[0.13951546 0.4        0.         0.33333333 1.         0.
 0.         0.         0.         0.33333333 1.         0.24545455]
[0.55444166 0.6        0.5        0.33333333 1.         1.
 0.         0.         0.         0.66666667 0.         0.3030303 ]
[0.12559176 0.2        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.15757576]
[0.11723754 0.2        0.5        0.         1.         0.
 1.         0.         0.         0.         0.         0.16363636]
[0.79114453 0.4        0.         0.33333333 1.         0.
 1.         0.         1.         0.66666667 1.         0.6969697 ]
[0.06015038 0.2        0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.1030303 ]
[0.17153996 1.         0.5        0.33333333 1.         0.
 0.         0.         0.         0.         0.         0.37515152]
[0.18546366 0.2        0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.13030303]
[0.21992481 0.6        0.5        0.         1.         0.
 1.         0.         0.         0.         0.         0.27212121]
[0.0858396  0.2        0.         0.         1.         0.
 0.         0.         0.         0.33333333 0.         0.12121212]
[0.17516012 0.6        0.         0.33333333 0.         0.
 0.         0.         0.         0.33333333 0.         0.10606061]
[0.49039265 0.4        0.5        0.33333333 1.         0.
 1.         0.         1.         0.33333333 0.         0.44848485]
[0.77332219 0.4        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.15151515]
[0.37064884 0.4        0.         0.         1.         0.
 1.         0.         0.         0.66666667 1.         0.35757576]]
```

[9]:
```
X_train = df_twoBtrain.values[:,0:11]
Y_train = df_twoBtrain.values[:,11]
len(X_train), len(Y_train)
```

[9]:
```
(436, 436)
```

[10]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[0.12419939 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         1.         0.
```

```
 0.         0.         0.         0.         0.        ]
[0.08382066 0.2        0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[0.12280702 0.2        0.         0.         1.         0.
 0.         0.         0.         0.         0.        ]
[0.55861877 0.4        0.         0.         1.         0.
 0.         0.         0.         0.         0.        ]]
Y = [0.         0.08181818 0.0969697  0.07272727 0.23939394]
```

[11]:
```
# Convert to 2D array (381x11)
m = len(X_train)
X_1 = X_train.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.12419939 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.08382066 0.2        0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.12280702 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.55861877 0.4        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]]
```

[12]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[12]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[13]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[13]:
```
array([[1.        , 0.12419939, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [1.        , 0.15065441, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [1.        , 0.08382066, 0.2       , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
```

```
       [1.        , 0.12280702, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [1.        , 0.55861877, 0.4       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ]])
```

[14]:
```
theta = np.zeros((12,1))
theta
```

[14]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[15]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[16]:
```
cost = compute_loss(X_train, Y_train, theta)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 20.934727519081726
```

[17]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[18]:
```
theta = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[19]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [0.07188371 0.05162113 0.05106637 0.06668738 0.0668938
0.07238152
 0.03894934 0.0359423  0.02096703 0.08066473 0.05619328 0.06046716]
loss_history = [0.0474144  0.04682244 0.04623943 … 0.00623665 0.00623586
0.00623508]
```

[20]:
```
df_twoBtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
239	4000	3	1	2	1	0	0	0	0	1	0	4585000
113	9620	3	1	1	1	0	1	0	0	2	1	6083000
325	3460	4	1	2	1	0	0	0	1	0	0	4007500
66	13200	2	1	1	1	0	1	1	0	1	0	6930000
479	3660	4	1	2	0	0	0	0	0	0	0	2940000

[21]:
```
# Many columns contains small integer values excluding areas. Needs to rescale
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_twoBtest[num_vars] = scaler.fit_transform(df_twoBtest[num_vars])
df_twoBtest.head()
```

[21]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
239	0.203463	0.50	0.0	0.333333	1.0	0.0	0.0	0.0	0.0	0.333333	0.0	0.270000
113	0.690043	0.50	0.0	0.000000	1.0	0.0	1.0	0.0	0.0	0.666667	1.0	0.412667
325	0.156710	0.75	0.0	0.333333	1.0	0.0	0.0	0.0	1.0	0.000000	0.0	0.215000
66	1.000000	0.25	0.0	0.000000	1.0	0.0	1.0	1.0	0.0	0.333333	0.0	0.493333
479	0.174026	0.75	0.0	0.333333	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.113333

[22]:
```
dataset_test = df_twoBtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[0.2034632  0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.33333333 0.         0.27      ]
 [0.69004329 0.5        0.         0.         1.         0.
  1.         0.         0.         0.66666667 1.         0.41266667]
 [0.15670996 0.75       0.         0.33333333 1.         0.
  0.         0.         1.         0.         0.         0.215     ]
 [1.         0.25       0.         0.         1.         0.
  1.         1.         0.         0.33333333 0.         0.49333333]
 [0.17402597 0.75       0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.         0.11333333]
 [0.40692641 0.5        0.33333333 0.66666667 1.         1.
  0.         0.         1.         0.         0.         0.42333333]
 [0.19047619 0.5        0.         0.         1.         0.
  0.         0.         0.         0.66666667 0.         0.17      ]
 [0.15844156 0.5        0.         0.33333333 0.         0.
  0.         0.         0.         0.33333333 0.         0.11333333]
 [0.16121212 0.25       0.         0.         1.         0.
  0.         0.         0.         0.33333333 1.         0.16666667]
 [0.63636364 0.75       0.33333333 1.         1.         0.
  0.         0.         1.         0.66666667 0.         0.59333333]
 [0.37662338 0.75       0.33333333 1.         1.         0.
  0.         0.         1.         0.         0.         0.47666667]
 [0.2        0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.         0.         0.21333333]
 [0.15584416 0.5        0.         0.33333333 1.         0.
  1.         0.         0.         0.         0.         0.13333333]
 [0.38181818 0.5        0.         0.         1.         1.
  1.         0.         0.         0.         0.         0.14666667]
 [0.37532468 0.5        0.         0.         1.         0.
  1.         0.         0.         0.         0.         0.22666667]
 [0.06753247 0.5        0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.08666667]
 [0.28138528 0.25       0.         0.33333333 1.         0.
  1.         0.         0.         0.         0.         0.25333333]
 [0.37835498 0.5        0.         0.         1.         0.
  0.         0.         0.         0.         0.         0.18666667]
 [0.12554113 0.5        0.         0.33333333 0.         0.
  1.         0.         0.         0.         0.         0.15333333]
 [0.24675325 0.25       0.         0.         1.         0.
  0.         0.         1.         0.66666667 0.         0.24      ]]
```

[23]:
```
X_test = df_twoBtest.values[:,0:11]
Y_test = df_twoBtest.values[:,11]
len(X_test), len(Y_test)
```

[23]: (109, 109)

[24]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[0.2034632  0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.33333333 0.        ]
 [0.69004329 0.5        0.         0.         1.         0.
  1.         0.         0.         0.66666667 1.        ]
 [0.15670996 0.75       0.         0.33333333 1.         0.
  0.         0.         1.         0.         0.        ]
 [1.         0.25       0.         0.         1.         0.
  1.         1.         0.         0.33333333 0.        ]
 [0.17402597 0.75       0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.        ]]
Y = [0.27       0.41266667 0.215      0.49333333 0.11333333]
```

[25]:
```
# Convert to 2D array (164x11)
m = len(X_test)
X_1 = X_test.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.2034632  0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.33333333 0.        ]
 [0.69004329 0.5        0.         0.         1.         0.
  1.         0.         0.         0.66666667 1.        ]
 [0.15670996 0.75       0.         0.33333333 1.         0.
  0.         0.         1.         0.         0.        ]
 [1.         0.25       0.         0.         1.         0.
  1.         1.         0.         0.33333333 0.        ]
 [0.17402597 0.75       0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.        ]]
```

[26]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[26]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[27]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[27]:
```
array([[1.        , 0.2034632 , 0.5       , 0.        , 0.33333333,
        1.        , 0.        , 0.        , 0.        , 0.        ,
```

```
 0.33333333, 0.        ],
[1.        , 0.69004329, 0.5       , 0.        , 0.        ,
 1.        , 0.        , 1.        , 0.        , 0.        ,
 0.66666667, 1.        ],
[1.        , 0.15670996, 0.75      , 0.        , 0.33333333,
 1.        , 0.        , 0.        , 0.        , 1.        ,
 0.        , 0.        ],
[1.        , 1.        , 0.25      , 0.        , 0.        ,
 1.        , 0.        , 1.        , 1.        , 0.        ,
 0.33333333, 0.        ],
[1.        , 0.17402597, 0.75      , 0.        , 0.33333333,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        ]])
```

[28]:
```
theta_test = np.zeros((12,1))
theta_test
```

[28]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[29]:
```
cost_test = compute_loss(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 5.79399238888889
```

[30]:
```
theta_test = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

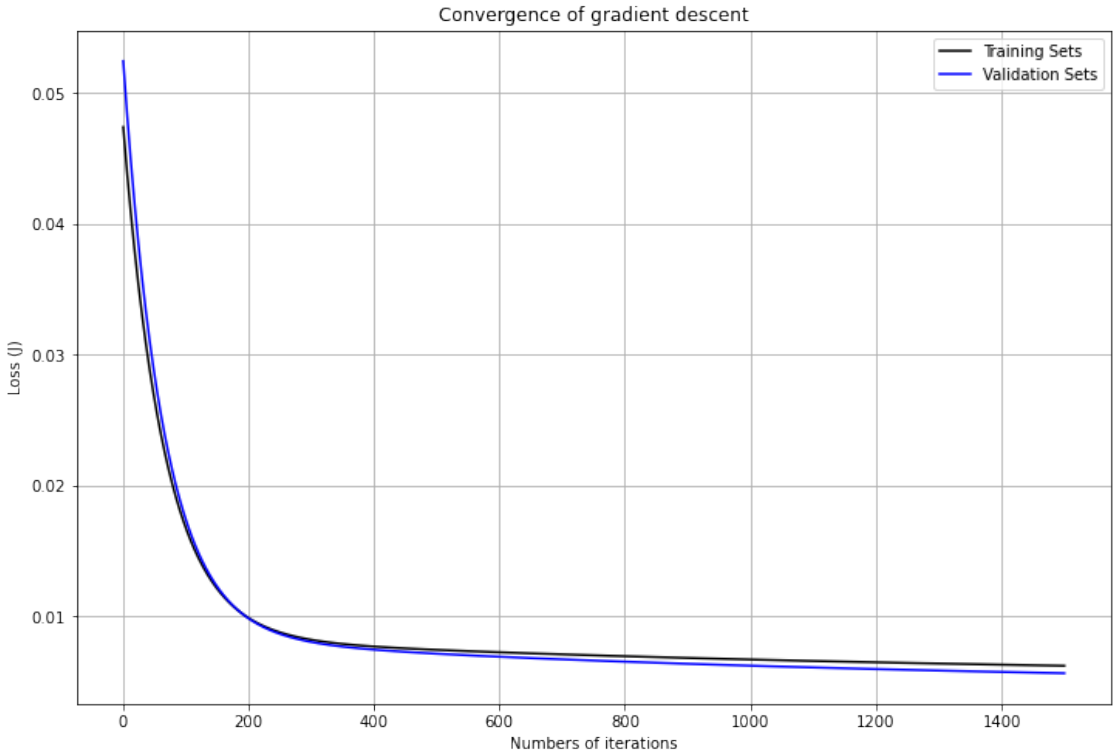
[31]:
```
theta_test, loss_history_test = gradient_descent(X_test, Y_test, theta_test,␣
 ↪alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [0.08068021 0.07227611 0.06077027 0.06066932 0.07529487
0.08690136
 0.0128184  0.02678734 0.00876003 0.0921445  0.06943986 0.01541427]
loss_history = [0.05245705 0.05176907 0.05109178 … 0.00566679 0.00566585
```

```
0.0056649 ]
```

[33]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[33]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]:
```
```

Problem_2b_Standardization

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Any dataset that has columns with values as 'Yes' or 'No', strings' values␣
 ↪cannot be used.
# However, we can convert them to numerical values as binary.

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished

[4]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[4]: (436, 13)

[5]:
```
df_test.shape
```

[5]: (109, 13)

[6]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',␣
 ↪'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',␣
 ↪'prefarea', 'price']
df_twoBtrain = df_train[num_vars]
df_twoBtest = df_test[num_vars]
df_twoBtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
542	3620	2	1	1	1	0	0	0	0	0	0	1750000
496	4000	2	1	1	1	0	0	0	0	0	0	2660000
484	3040	2	1	1	0	0	0	0	0	0	0	2800000
507	3600	2	1	1	1	0	0	0	0	0	0	2450000

```
252  9860         3          1        1         1          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
542                0                0        0         0  1750000
496                0                0        0         0  2695000
484                0                0        0         0  2870000
507                0                0        0         0  2590000
252                0                0        0         0  4515000
```

[7]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_twoBtrain[num_vars] = scaler.fit_transform(df_twoBtrain[num_vars])
df_twoBtrain.head()
```

[7]:
```
         area  bedrooms  bathrooms   stories  mainroad  guestroom  basement  \
542 -0.716772 -1.294376  -0.573307 -0.933142  0.395599  -0.463125 -0.698609
496 -0.538936 -1.294376  -0.573307 -0.933142  0.395599  -0.463125 -0.698609
484 -0.988206 -1.294376  -0.573307 -0.933142 -2.527811  -0.463125 -0.698609
507 -0.726132 -1.294376  -0.573307 -0.933142  0.395599  -0.463125 -0.698609
252  2.203478  0.052516  -0.573307 -0.933142  0.395599  -0.463125 -0.698609

     hotwaterheating  airconditioning   parking  prefarea     price
542        -0.201427        -0.691351 -0.819149 -0.570288 -1.586001
496        -0.201427        -0.691351 -0.819149 -0.570288 -1.090971
484        -0.201427        -0.691351 -0.819149 -0.570288 -0.999299
507        -0.201427        -0.691351 -0.819149 -0.570288 -1.145974
252        -0.201427        -0.691351 -0.819149 -0.570288 -0.137579
```

[8]:
```
dataset_train = df_twoBtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[-0.71677205 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -1.5860012 ]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -1.09097091]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
```

```
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.99929863]
[-0.72613182 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -1.14597428]
[ 2.20347795  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.13757923]
[-0.55391195  0.05251643 -0.57330726  0.21291401 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.1925826 ]
[-0.61381451  0.05251643 -0.57330726  0.21291401  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093  0.32555914  1.75350117 -0.10091032]
[ 2.17539862  1.39940847  1.4755613   0.21291401  0.39559913  2.1592447
 -0.69860905 -0.20142689 -0.69135093  1.47026706 -0.57028761  0.24744433]
[-0.70741227 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.63260953]
[-0.76357092 -1.29437561  1.4755613  -0.93314164  0.39559913 -0.46312491
  1.43141575 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.59594061]
[ 3.76656047  0.05251643 -0.57330726  0.21291401  0.39559913 -0.46312491
  1.43141575 -0.20142689  1.44644342  1.47026706  1.75350117  2.63092353]
[-1.14732172 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.96262972]
[-0.39853968  4.09319255  1.4755613   0.21291401  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761  0.68380437]
[-0.30494192 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.79761962]
[-0.07328747  1.39940847  1.4755613  -0.93314164  0.39559913 -0.46312491
  1.43141575 -0.20142689 -0.69135093 -0.81914879 -0.57028761  0.06043289]
[-0.97463386 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093  0.32555914 -0.57028761 -0.85262299]
[-0.37420426  1.39940847 -0.57330726  0.21291401 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093  0.32555914 -0.57028761 -0.94429527]
[ 1.74484894  0.05251643  1.4755613   0.21291401  0.39559913 -0.46312491
  1.43141575 -0.20142689  1.44644342  0.32555914 -0.57028761  1.12749819]
[ 3.64675535  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.66927844]
[ 0.93990824  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
  1.43141575 -0.20142689 -0.69135093  1.47026706  1.75350117  0.57746453]]
```

[9]:
```
X_train = df_twoBtrain.values[:,0:11]
Y_train = df_twoBtrain.values[:,11]
len(X_train), len(Y_train)
```

[9]:
```
(436, 436)
```

[10]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
```

```
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
[-0.98820554 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
[-0.72613182 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
[ 2.20347795  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]]
Y = [-1.5860012  -1.09097091 -0.99929863 -1.14597428 -0.13757923]
```

[11]:
```
# Convert to 2D array (381x11)
m = len(X_train)
X_1 = X_train.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]]
```

[12]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[12]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[13]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[13]:
```
array([[ 1.        , -0.71677205, -1.29437561, -0.57330726, -0.93314164,
         0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
       [ 1.        , -0.53893631, -1.29437561, -0.57330726, -0.93314164,
         0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
       [ 1.        , -0.98820554, -1.29437561, -0.57330726, -0.93314164,
        -2.52781141, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
```

```
       [ 1.        , -0.72613182, -1.29437561, -0.57330726, -0.93314164,
         0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
       [ 1.        ,  2.20347795,  0.05251643, -0.57330726, -0.93314164,
         0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761]])
```

[14]:
```
theta = np.zeros((12,1))
theta
```

[14]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[15]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[16]:
```
cost = compute_loss(X_train, Y_train, theta)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 218.00000000000006
```

[17]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta)
    return theta, loss_history
```

[18]:
```
theta = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[19]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [2.53191963e-16 2.79680711e-01 6.76860944e-02
2.56953948e-01
 1.93641900e-01 8.90891937e-02 9.11258033e-02 8.23417050e-02
 1.22587049e-01 2.18754352e-01 1.16428408e-01 1.61503921e-01]
loss_history = [0.49533842 0.49074342 0.48621404 … 0.16423687 0.16423662
0.16423637]
```

[20]:
```
df_twoBtest.head()
```

[20]:
```
      area  bedrooms  bathrooms  stories  mainroad  guestroom  basement  \
239   4000         3          1        2         1          0         0
113   9620         3          1        1         1          0         1
325   3460         4          1        2         1          0         0
66   13200         2          1        1         1          0         1
479   3660         4          1        2         0          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
239                0                0        1         0  4585000
113                0                0        2         1  6083000
325                0                1        0         0  4007500
66                 1                0        1         0  6930000
479                0                0        0         0  2940000
```

[21]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_twoBtest[num_vars] = scaler.fit_transform(df_twoBtest[num_vars])
df_twoBtest.head()
```

[21]:
```
          area  bedrooms  bathrooms   stories  mainroad  guestroom  basement  \
239 -0.500735  0.025607  -0.563545  0.272416  0.444750  -0.474045 -0.887066
113  1.954229  0.025607  -0.563545 -0.915317  0.444750  -0.474045  1.127312
325 -0.736621  1.421209  -0.563545  0.272416  0.444750  -0.474045 -0.887066
66   3.518067 -1.369995  -0.563545 -0.915317  0.444750  -0.474045  1.127312
479 -0.649256  1.421209  -0.563545  0.272416 -2.248456  -0.474045 -0.887066

     hotwaterheating  airconditioning   parking  prefarea     price
239        -0.281439        -0.630425  0.492144 -0.488504 -0.081358
113        -0.281439        -0.630425  1.739673  2.047065  0.801114
325        -0.281439         1.586231 -0.755384 -0.488504 -0.421563
66          3.553168        -0.630425  0.492144 -0.488504  1.300082
479        -0.281439        -0.630425 -0.755384 -0.488504 -1.050428
```

[22]:
```
dataset_test = df_twoBtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[-0.50073521  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421 -0.08135801]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517  1.73967255  2.04706526  0.80111439]
 [-0.73662142  1.42120937 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421 -0.42156349]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
    1.12731244  3.5531676  -0.63042517  0.49214421 -0.48850421  1.30008243]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -1.05042817]
 [ 0.52580664  0.02560738  1.2431129   1.46014902  0.44474959  2.10950231
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421  0.86709364]
 [-0.56625916  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  1.73967255 -0.48850421 -0.69991343]
 [-0.72788489  0.02560738 -0.56354451  0.27241586 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421 -1.05042817]
 [-0.71390645 -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421  2.04706526 -0.72053194]
 [ 1.68339637  1.42120937  1.2431129   2.64788217  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108  1.73967255 -0.48850421  1.91863786]
 [ 0.37291742  1.42120937  1.2431129   2.64788217  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421  1.19698986]
 [-0.51820826  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.43187275]
 [-0.74098968  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.92671708]
 [ 0.399127    0.02560738 -0.56354451 -0.91531729  0.44474959  2.10950231
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.84424303]
 [ 0.36636503  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.34939869]
 [-1.18655253  0.02560738 -0.56354451 -0.91531729 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -1.21537628]
 [-0.10759152 -1.36999462 -0.56354451  0.27241586  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.18445058]
 [ 0.38165395  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.59682086]
 [-0.89387889  0.02560738 -0.56354451  0.27241586 -2.24845626 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.803006  ]
 [-0.28232205 -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108  1.73967255 -0.48850421 -0.26692463]]
```

[23]:
```
X_test = df_twoBtest.values[:,0:11]
Y_test = df_twoBtest.values[:,11]
len(X_test), len(Y_test)
```

[23]: (109, 109)

[24]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244 -0.28143902 -0.63042517  1.73967255  2.04706526]
 [-0.73662142  1.42120937 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244  3.5531676  -0.63042517  0.49214421 -0.48850421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421]]
Y = [-0.08135801  0.80111439 -0.42156349  1.30008243 -1.05042817]
```

[25]:
```
# Convert to 2D array (164x11)
m = len(X_test)
X_1 = X_test.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244 -0.28143902 -0.63042517  1.73967255  2.04706526]
 [-0.73662142  1.42120937 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244  3.5531676  -0.63042517  0.49214421 -0.48850421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421]]
```

[26]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[26]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[27]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[27]:
```
array([[ 1.        , -0.50073521,  0.02560738, -0.56354451,  0.27241586,
         0.44474959, -0.47404546, -0.88706553, -0.28143902, -0.63042517,
```

```
  0.49214421, -0.48850421],
[ 1.        ,  1.95422869,  0.02560738, -0.56354451, -0.91531729,
  0.44474959, -0.47404546,  1.12731244, -0.28143902, -0.63042517,
  1.73967255,  2.04706526],
[ 1.        , -0.73662142,  1.42120937, -0.56354451,  0.27241586,
  0.44474959, -0.47404546, -0.88706553, -0.28143902,  1.58623108,
 -0.75538413, -0.48850421],
[ 1.        ,  3.5180669 , -1.36999462, -0.56354451, -0.91531729,
  0.44474959, -0.47404546,  1.12731244,  3.5531676 , -0.63042517,
  0.49214421, -0.48850421],
[ 1.        , -0.64925616,  1.42120937, -0.56354451,  0.27241586,
 -2.24845626, -0.47404546, -0.88706553, -0.28143902, -0.63042517,
 -0.75538413, -0.48850421]])
```

[28]:
```
theta_test = np.zeros((12,1))
theta_test
```

[28]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[29]:
```
cost_test = compute_loss(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 54.49999999999999
```

[30]:
```
theta_test = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

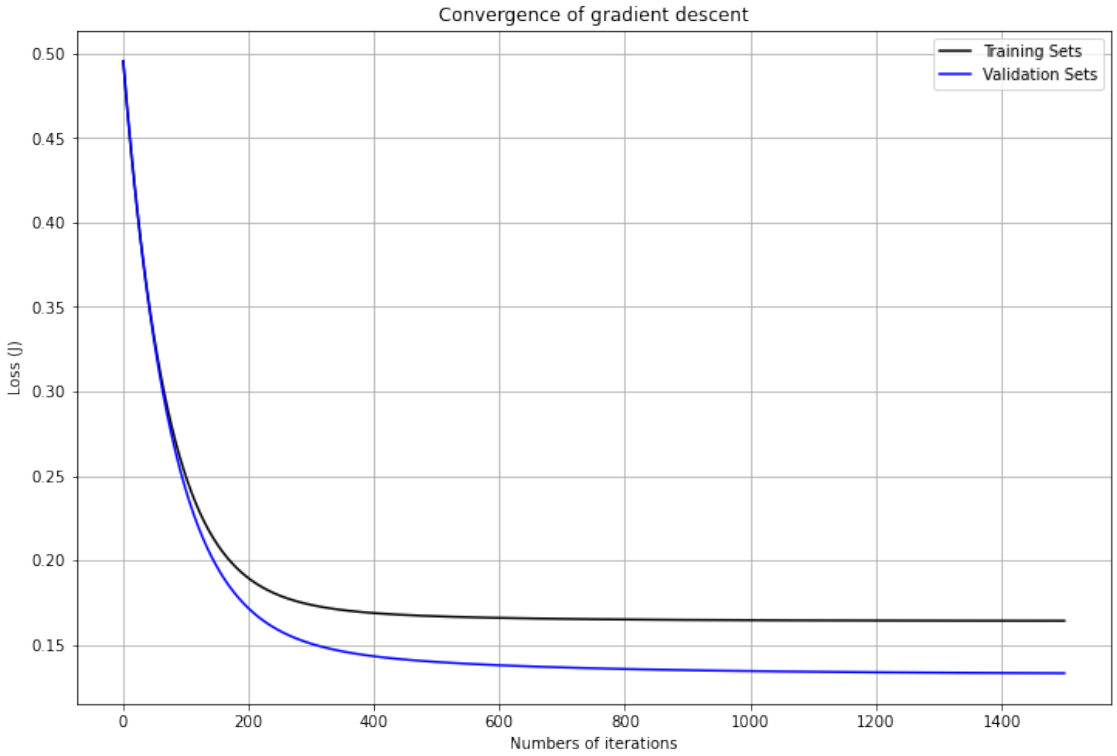
[31]:
```
theta_test, loss_history_test = gradient_descent(X_test, Y_test, theta_test,␣
 ↪alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [ 1.27995474e-16  2.70877429e-01  1.98422728e-02
3.05759749e-01
  2.31850131e-01  1.28966576e-01 -4.03787210e-02  1.66329173e-01
  4.35598057e-02  2.61782180e-01  2.34913087e-01  5.10329844e-02]
```

```
loss_history = [0.49538399 0.49082909 0.48633448 … 0.13320429 0.13320306
0.13320183]
```

[33]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[33]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]:

Problem_3a_MinMax

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[3]: (436, 13)

[4]:
```
df_test.shape
```

[4]: (109, 13)

[5]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
df_threeAtrain = df_train[num_vars]
```

```
df_threeAtest = df_test[num_vars]
df_threeAtrain.head()
```

[5]:

	area	bedrooms	bathrooms	stories	parking	price
542	3620	2	1	1	0	1750000
496	4000	2	1	1	0	2695000
484	3040	2	1	1	0	2870000
507	3600	2	1	1	0	2590000
252	9860	3	1	1	0	4515000

[6]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_threeAtrain[num_vars] = scaler.fit_transform(df_threeAtrain[num_vars])
df_threeAtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	parking	price
542	0.124199	0.2	0.0	0.0	0.0	0.000000
496	0.150654	0.2	0.0	0.0	0.0	0.081818
484	0.083821	0.2	0.0	0.0	0.0	0.096970
507	0.122807	0.2	0.0	0.0	0.0	0.072727
252	0.558619	0.4	0.0	0.0	0.0	0.239394

[7]:
```
dataset_train = df_threeAtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[0.12419939 0.2        0.         0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         0.         0.08181818]
 [0.08382066 0.2        0.         0.         0.         0.0969697 ]
 [0.12280702 0.2        0.         0.         0.         0.07272727]
 [0.55861877 0.4        0.         0.         0.         0.23939394]
 [0.14842662 0.4        0.         0.33333333 0.         0.23030303]
 [0.13951546 0.4        0.         0.33333333 0.33333333 0.24545455]
 [0.55444166 0.6        0.5        0.33333333 0.66666667 0.3030303 ]
 [0.12559176 0.2        0.         0.         0.         0.15757576]
 [0.11723754 0.2        0.5        0.         0.         0.16363636]
 [0.79114453 0.4        0.         0.33333333 0.66666667 0.6969697 ]
```

```
 [0.06015038 0.2        0.         0.         0.         0.1030303 ]
 [0.17153996 1.         0.5        0.33333333 0.         0.37515152]
 [0.18546366 0.2        0.         0.         0.         0.13030303]
 [0.21992481 0.6        0.5        0.         0.         0.27212121]
 [0.0858396  0.2        0.         0.         0.33333333 0.12121212]
 [0.17516012 0.6        0.         0.33333333 0.33333333 0.10606061]
 [0.49039265 0.4        0.5        0.33333333 0.33333333 0.44848485]
 [0.77332219 0.4        0.         0.         0.         0.15151515]
 [0.37064884 0.4        0.         0.         0.66666667 0.35757576]]
```

[8]:
```
X_train = df_threeAtrain.values[:,0:5]
Y_train = df_threeAtrain.values[:,5]
len(X_train), len(Y_train)
```

[8]:
```
(436, 436)
```

[9]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[0.12419939 0.2        0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         0.        ]
 [0.08382066 0.2        0.         0.         0.        ]
 [0.12280702 0.2        0.         0.         0.        ]
 [0.55861877 0.4        0.         0.         0.        ]]
Y = [0.         0.08181818 0.0969697  0.07272727 0.23939394]
```

[10]:
```
# Convert to 2D array (381x5)
m = len(X_train)
X_1 = X_train.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.12419939 0.2        0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         0.        ]
 [0.08382066 0.2        0.         0.         0.        ]
 [0.12280702 0.2        0.         0.         0.        ]
 [0.55861877 0.4        0.         0.         0.        ]]
```

[11]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[11]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[12]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[12]:
```
array([[1.        , 0.12419939, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.15065441, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.08382066, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.12280702, 0.2       , 0.        , 0.        ,
        0.        ],
       [1.        , 0.55861877, 0.4       , 0.        , 0.        ,
        0.        ]])
```

[13]:
```
theta = np.zeros((6,1))
theta
```

[13]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[14]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m
lambda_value: Regularization parameter.
theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta, lambda_value):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    regularization = lambda_value * np.sum(np.square(theta))
    J = (1 / (2 * m)) * (np.sum(sqrErrors) + regularization)
    return J
```

[15]:
```
lambda_value = 10
cost = compute_loss(X_train, Y_train, theta, lambda_value)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 20.934727519081726
```

[16]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.
lambda_value: Regularization parameter.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations, lambda_value):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        # Use np.multiply() to multiple scalar with the array, theta.
        sum_delta = (alpha / m) * X.transpose().dot(errors) + np.
 ↪multiply(theta,((lambda_value * alpha) / m));
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta, lambda_value)
    return theta, loss_history
```

[17]:
```
theta = [0., 0., 0., 0., 0., 0.]
lambda_value = 10
iterations = 1500
alpha = 0.003
```

[18]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations, lambda_value)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [0.15720441 0.07938084 0.08491886 0.08054867 0.09057556
```

```
0.08264286]
loss_history = [0.0477018  0.0473908  0.0470824  … 0.00929665 0.009296
0.00929535]
```

[19]:
```
df_threeAtest.head()
```

[19]:

	area	bedrooms	bathrooms	stories	parking	price
239	4000	3	1	2	1	4585000
113	9620	3	1	1	2	6083000
325	3460	4	1	2	0	4007500
66	13200	2	1	1	1	6930000
479	3660	4	1	2	0	2940000

[20]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_threeAtest[num_vars] = scaler.fit_transform(df_threeAtest[num_vars])
df_threeAtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	parking	price
239	0.203463	0.50	0.0	0.333333	0.333333	0.270000
113	0.690043	0.50	0.0	0.000000	0.666667	0.412667
325	0.156710	0.75	0.0	0.333333	0.000000	0.215000
66	1.000000	0.25	0.0	0.000000	0.333333	0.493333
479	0.174026	0.75	0.0	0.333333	0.000000	0.113333

[21]:
```
dataset_test = df_threeAtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[0.2034632  0.5        0.         0.33333333 0.33333333 0.27      ]
 [0.69004329 0.5        0.         0.         0.66666667 0.41266667]
 [0.15670996 0.75       0.         0.33333333 0.         0.215     ]
 [1.         0.25       0.         0.         0.33333333 0.49333333]
 [0.17402597 0.75       0.         0.33333333 0.         0.11333333]
 [0.40692641 0.5        0.33333333 0.66666667 0.         0.42333333]
 [0.19047619 0.5        0.         0.         0.66666667 0.17      ]
 [0.15844156 0.5        0.         0.33333333 0.33333333 0.11333333]
```

```
 [0.16121212 0.25       0.         0.         0.33333333 0.16666667]
 [0.63636364 0.75       0.33333333 1.         0.66666667 0.59333333]
 [0.37662338 0.75       0.33333333 1.         0.         0.47666667]
 [0.2        0.5        0.         0.33333333 0.         0.21333333]
 [0.15584416 0.5        0.         0.33333333 0.         0.13333333]
 [0.38181818 0.5        0.         0.         0.         0.14666667]
 [0.37532468 0.5        0.         0.         0.         0.22666667]
 [0.06753247 0.5        0.         0.         0.         0.08666667]
 [0.28138528 0.25       0.         0.33333333 0.         0.25333333]
 [0.37835498 0.5        0.         0.         0.         0.18666667]
 [0.12554113 0.5        0.         0.33333333 0.         0.15333333]
 [0.24675325 0.25       0.         0.         0.66666667 0.24      ]]
```

[22]:
```
X_test = df_threeAtest.values[:,0:5]
Y_test = df_threeAtest.values[:,5]
len(X_test), len(Y_test)
```

[22]:
```
(109, 109)
```

[23]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[0.2034632  0.5        0.         0.33333333 0.33333333]
 [0.69004329 0.5        0.         0.         0.66666667]
 [0.15670996 0.75       0.         0.33333333 0.        ]
 [1.         0.25       0.         0.         0.33333333]
 [0.17402597 0.75       0.         0.33333333 0.        ]]
Y = [0.27       0.41266667 0.215      0.49333333 0.11333333]
```

[24]:
```
# Convert to 2D array (164x5)
m = len(X_test)
X_1 = X_test.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.2034632  0.5        0.         0.33333333 0.33333333]
 [0.69004329 0.5        0.         0.         0.66666667]
 [0.15670996 0.75       0.         0.33333333 0.        ]
 [1.         0.25       0.         0.         0.33333333]
 [0.17402597 0.75       0.         0.33333333 0.        ]]
```

[25]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[25]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
```

```
        [1.]]),
 109)
```

[26]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[26]:
```
array([[1.        , 0.2034632 , 0.5       , 0.        , 0.33333333,
        0.33333333],
       [1.        , 0.69004329, 0.5       , 0.        , 0.        ,
        0.66666667],
       [1.        , 0.15670996, 0.75      , 0.        , 0.33333333,
        0.        ],
       [1.        , 1.        , 0.25      , 0.        , 0.        ,
        0.33333333],
       [1.        , 0.17402597, 0.75      , 0.        , 0.33333333,
        0.        ]])
```

[27]:
```
theta_test = np.zeros((6,1))
theta_test
```

[27]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[28]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss_noreg(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
```

```
    return J
```

[29]:
```
cost_test = compute_loss_noreg(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 5.79399238888889
```

[30]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent_noreg(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss_noreg(X, Y, theta)
    return theta, loss_history
```

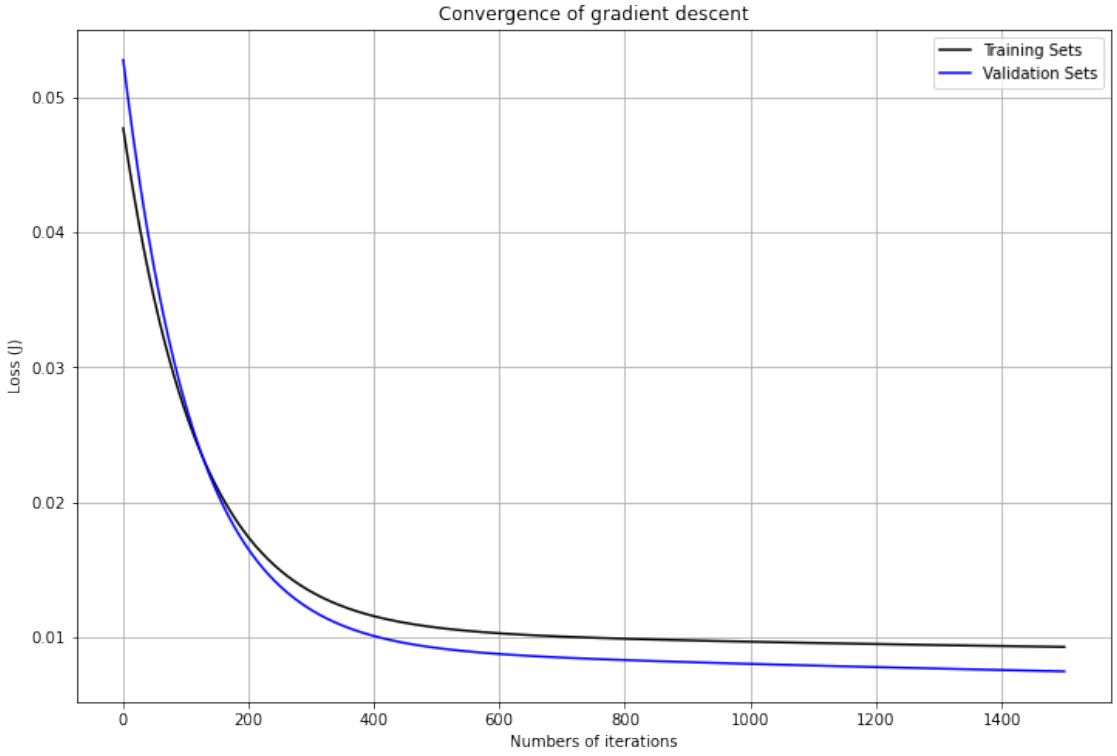
[31]:
```
theta_test = [0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[32]:
```
theta_test, loss_history_test = gradient_descent_noreg(X_test, Y_test,␣
 ↪theta_test, alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [0.161425   0.10594748 0.09248631 0.07041321 0.09856776
0.08675523]
loss_history = [0.05276804 0.0523836  0.05200255 … 0.00748962 0.00748864
0.00748766]
```

[34]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[34]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]:

Problem_3a_Standardization

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[3]: (436, 13)

[4]:
```
df_test.shape
```

[4]: (109, 13)

[5]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'price']
df_threeAtrain = df_train[num_vars]
```

```
df_threeAtest = df_test[num_vars]
df_threeAtrain.head()
```

[5]:

	area	bedrooms	bathrooms	stories	parking	price
542	3620	2	1	1	0	1750000
496	4000	2	1	1	0	2695000
484	3040	2	1	1	0	2870000
507	3600	2	1	1	0	2590000
252	9860	3	1	1	0	4515000

[6]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_threeAtrain[num_vars] = scaler.fit_transform(df_threeAtrain[num_vars])
df_threeAtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	parking	price
542	-0.716772	-1.294376	-0.573307	-0.933142	-0.819149	-1.586001
496	-0.538936	-1.294376	-0.573307	-0.933142	-0.819149	-1.090971
484	-0.988206	-1.294376	-0.573307	-0.933142	-0.819149	-0.999299
507	-0.726132	-1.294376	-0.573307	-0.933142	-0.819149	-1.145974
252	2.203478	0.052516	-0.573307	-0.933142	-0.819149	-0.137579

[7]:
```
dataset_train = df_threeAtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[-0.71677205 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -1.5860012 ]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -1.09097091]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.99929863]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -1.14597428]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164 -0.81914879 -0.13757923]
 [-0.55391195  0.05251643 -0.57330726  0.21291401 -0.81914879 -0.1925826 ]
 [-0.61381451  0.05251643 -0.57330726  0.21291401  0.32555914 -0.10091032]
 [ 2.17539862  1.39940847  1.4755613   0.21291401  1.47026706  0.24744433]
 [-0.70741227 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.63260953]
 [-0.76357092 -1.29437561  1.4755613  -0.93314164 -0.81914879 -0.59594061]
 [ 3.76656047  0.05251643 -0.57330726  0.21291401  1.47026706  2.63092353]
```

```
 [-1.14732172 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.96262972]
 [-0.39853968  4.09319255  1.4755613   0.21291401 -0.81914879  0.68380437]
 [-0.30494192 -1.29437561 -0.57330726 -0.93314164 -0.81914879 -0.79761962]
 [-0.07328747  1.39940847  1.4755613  -0.93314164 -0.81914879  0.06043289]
 [-0.97463386 -1.29437561 -0.57330726 -0.93314164  0.32555914 -0.85262299]
 [-0.37420426  1.39940847 -0.57330726  0.21291401  0.32555914 -0.94429527]
 [ 1.74484894  0.05251643  1.4755613   0.21291401  0.32555914  1.12749819]
 [ 3.64675535  0.05251643 -0.57330726 -0.93314164 -0.81914879 -0.66927844]
 [ 0.93990824  0.05251643 -0.57330726 -0.93314164  1.47026706  0.57746453]]
```

[8]:
```
X_train = df_threeAtrain.values[:,0:5]
Y_train = df_threeAtrain.values[:,5]
len(X_train), len(Y_train)
```

[8]:
```
(436, 436)
```

[9]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164 -0.81914879]]
Y = [-1.5860012  -1.09097091 -0.99929863 -1.14597428 -0.13757923]
```

[10]:
```
# Convert to 2D array (381x5)
m = len(X_train)
X_1 = X_train.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164 -0.81914879]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164 -0.81914879]]
```

[11]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[11]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[12]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[12]:
```
array([[ 1.        , -0.71677205, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        , -0.53893631, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        , -0.98820554, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        , -0.72613182, -1.29437561, -0.57330726, -0.93314164,
        -0.81914879],
       [ 1.        ,  2.20347795,  0.05251643, -0.57330726, -0.93314164,
        -0.81914879]])
```

[13]:
```
theta = np.zeros((6,1))
#lambda_value = 10
theta
```

[13]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[14]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m
lambda_value: Regularization parameter.
theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta, lambda_value):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    regularization = lambda_value * np.sum(np.square(theta))
    J = (1 / (2 * m)) * (np.sum(sqrErrors) + regularization)
    return J
```

[15]:
```
lambda_value = 10
cost = compute_loss(X_train, Y_train, theta, lambda_value)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 218.00000000000006
```

[16]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.
lambda_value: Regularization parameter.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations, lambda_value):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        # Use np.multiply() to multiple scalar with the array, theta.
        sum_delta = (alpha / m) * X.transpose().dot(errors) + np.
 ↪multiply(theta,((lambda_value * alpha) / m));
        theta = theta - sum_delta #- regularization; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta, lambda_value)
    return theta, loss_history
```

[17]:
```
theta = [0., 0., 0., 0., 0., 0.]
lambda_value = 10
iterations = 1500
alpha = 0.003
```

[18]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations, lambda_value)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [2.50556126e-16 3.74927030e-01 1.00969227e-01
```

```
2.94974466e-01
 2.31261970e-01 1.63921955e-01]
loss_history = [0.49701369 0.49406115 0.49114199 … 0.2269271  0.2269269
0.2269267 ]
```

[19]:
```
df_threeAtest.head()
```

[19]:

	area	bedrooms	bathrooms	stories	parking	price
239	4000	3	1	2	1	4585000
113	9620	3	1	1	2	6083000
325	3460	4	1	2	0	4007500
66	13200	2	1	1	1	6930000
479	3660	4	1	2	0	2940000

[20]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_threeAtest[num_vars] = scaler.fit_transform(df_threeAtest[num_vars])
df_threeAtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	parking	price
239	-0.500735	0.025607	-0.563545	0.272416	0.492144	-0.081358
113	1.954229	0.025607	-0.563545	-0.915317	1.739673	0.801114
325	-0.736621	1.421209	-0.563545	0.272416	-0.755384	-0.421563
66	3.518067	-1.369995	-0.563545	-0.915317	0.492144	1.300082
479	-0.649256	1.421209	-0.563545	0.272416	-0.755384	-1.050428

[21]:
```
dataset_test = df_threeAtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[-0.50073521  0.02560738 -0.56354451  0.27241586  0.49214421 -0.08135801]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  1.73967255  0.80111439]
 [-0.73662142  1.42120937 -0.56354451  0.27241586 -0.75538413 -0.42156349]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.49214421  1.30008243]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -0.75538413 -1.05042817]
 [ 0.52580664  0.02560738  1.2431129   1.46014902 -0.75538413  0.86709364]
 [-0.56625916  0.02560738 -0.56354451 -0.91531729  1.73967255 -0.69991343]
```

```
 [-0.72788489  0.02560738 -0.56354451  0.27241586  0.49214421 -1.05042817]
 [-0.71390645 -1.36999462 -0.56354451 -0.91531729  0.49214421 -0.72053194]
 [ 1.68339637  1.42120937  1.2431129   2.64788217  1.73967255  1.91863786]
 [ 0.37291742  1.42120937  1.2431129   2.64788217 -0.75538413  1.19698986]
 [-0.51820826  0.02560738 -0.56354451  0.27241586 -0.75538413 -0.43187275]
 [-0.74098968  0.02560738 -0.56354451  0.27241586 -0.75538413 -0.92671708]
 [ 0.399127    0.02560738 -0.56354451 -0.91531729 -0.75538413 -0.84424303]
 [ 0.36636503  0.02560738 -0.56354451 -0.91531729 -0.75538413 -0.34939869]
 [-1.18655253  0.02560738 -0.56354451 -0.91531729 -0.75538413 -1.21537628]
 [-0.10759152 -1.36999462 -0.56354451  0.27241586 -0.75538413 -0.18445058]
 [ 0.38165395  0.02560738 -0.56354451 -0.91531729 -0.75538413 -0.59682086]
 [-0.89387889  0.02560738 -0.56354451  0.27241586 -0.75538413 -0.803006  ]
 [-0.28232205 -1.36999462 -0.56354451 -0.91531729  1.73967255 -0.26692463]]
```

[22]:
```
X_test = df_threeAtest.values[:,0:5]
Y_test = df_threeAtest.values[:,5]
len(X_test), len(Y_test)
```

[22]:
```
(109, 109)
```

[23]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.49214421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  1.73967255]
 [-0.73662142  1.42120937 -0.56354451  0.27241586 -0.75538413]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.49214421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -0.75538413]]
Y = [-0.08135801  0.80111439 -0.42156349  1.30008243 -1.05042817]
```

[24]:
```
# Convert to 2D array (164x5)
m = len(X_test)
X_1 = X_test.reshape(m,5)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.49214421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  1.73967255]
 [-0.73662142  1.42120937 -0.56354451  0.27241586 -0.75538413]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.49214421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -0.75538413]]
```

[25]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[25]:
```
(array([[1.],
        [1.],
        [1.],
```

```
        [1.],
        [1.]]),
 109)
```

[26]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[26]:
```
array([[ 1.        , -0.50073521,  0.02560738, -0.56354451,  0.27241586,
         0.49214421],
       [ 1.        ,  1.95422869,  0.02560738, -0.56354451, -0.91531729,
         1.73967255],
       [ 1.        , -0.73662142,  1.42120937, -0.56354451,  0.27241586,
        -0.75538413],
       [ 1.        ,  3.5180669 , -1.36999462, -0.56354451, -0.91531729,
         0.49214421],
       [ 1.        , -0.64925616,  1.42120937, -0.56354451,  0.27241586,
        -0.75538413]])
```

[27]:
```
theta_test = np.zeros((6,1))
theta_test
```

[27]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[28]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss_noreg(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
```

```
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[29]:
```
cost_test = compute_loss_noreg(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 54.49999999999999
```

[30]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent_noreg(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss_noreg(X, Y, theta)
    return theta, loss_history
```

[31]:
```
theta_test = [0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

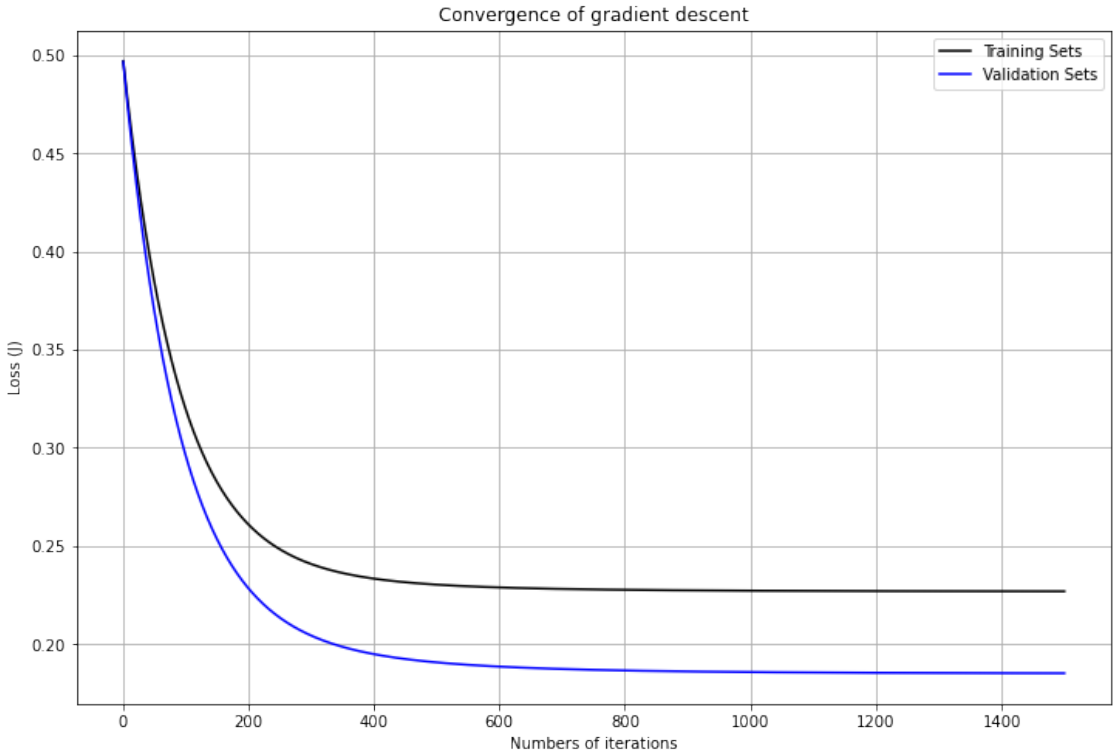
[32]:
```
theta_test, loss_history_test = gradient_descent_noreg(X_test, Y_test,␣
 ↪theta_test, alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [ 9.28431643e-17  3.58547244e-01 -3.04213335e-02
3.28020165e-01
  3.06752025e-01  2.34115221e-01]
```

```
loss_history = [0.49663315 0.49330422 0.49001277 … 0.18520046 0.18520007
0.18519967]
```

[34]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[34]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]:

Problem_3b_MinMax

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Any dataset that has columns with values as 'Yes' or 'No', strings' values␣
 ↪cannot be used.
# However, we can convert them to numerical values as binary.

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished

[4]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[4]:
```
(436, 13)
```

[5]:
```
df_test.shape
```

[5]:
```
(109, 13)
```

[6]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',␣
 ↪'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',␣
 ↪'prefarea', 'price']
df_threeBtrain = df_train[num_vars]
df_threeBtest = df_test[num_vars]
df_threeBtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
542	3620	2	1	1	1	0	0	0	0	0	0	1750000
496	4000	2	1	1	1	0	0	0	0	0	0	2660000
484	3040	2	1	1	0	0	0	0	0	0	0	2800000
507	3600	2	1	1	1	0	0	0	0	0	0	2450000

```
252  9860         3          1        1         1          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
542                0                0        0         0  1750000
496                0                0        0         0  2695000
484                0                0        0         0  2870000
507                0                0        0         0  2590000
252                0                0        0         0  4515000
```

[7]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_threeBtrain[num_vars] = scaler.fit_transform(df_threeBtrain[num_vars])
df_threeBtrain.head()
```

[7]:
```
         area  bedrooms  bathrooms  stories  mainroad  guestroom  basement  \
542  0.124199       0.2        0.0      0.0       1.0        0.0       0.0
496  0.150654       0.2        0.0      0.0       1.0        0.0       0.0
484  0.083821       0.2        0.0      0.0       0.0        0.0       0.0
507  0.122807       0.2        0.0      0.0       1.0        0.0       0.0
252  0.558619       0.4        0.0      0.0       1.0        0.0       0.0

     hotwaterheating  airconditioning  parking  prefarea     price
542              0.0              0.0      0.0       0.0  0.000000
496              0.0              0.0      0.0       0.0  0.081818
484              0.0              0.0      0.0       0.0  0.096970
507              0.0              0.0      0.0       0.0  0.072727
252              0.0              0.0      0.0       0.0  0.239394
```

[8]:
```
dataset_train = df_threeBtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[0.12419939 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.         ]
 [0.15065441 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.08181818]
 [0.08382066 0.2        0.         0.         0.         0.
```

```
 0.         0.         0.         0.         0.         0.0969697 ]
[0.12280702 0.2        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.07272727]
[0.55861877 0.4        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.23939394]
[0.14842662 0.4        0.         0.33333333 0.         0.
 0.         0.         0.         0.         0.         0.23030303]
[0.13951546 0.4        0.         0.33333333 1.         0.
 0.         0.         0.         0.33333333 1.         0.24545455]
[0.55444166 0.6        0.5        0.33333333 1.         1.
 0.         0.         0.         0.66666667 0.         0.3030303 ]
[0.12559176 0.2        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.15757576]
[0.11723754 0.2        0.5        0.         1.         0.
 1.         0.         0.         0.         0.         0.16363636]
[0.79114453 0.4        0.         0.33333333 1.         0.
 1.         0.         1.         0.66666667 1.         0.6969697 ]
[0.06015038 0.2        0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.1030303 ]
[0.17153996 1.         0.5        0.33333333 1.         0.
 0.         0.         0.         0.         0.         0.37515152]
[0.18546366 0.2        0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.13030303]
[0.21992481 0.6        0.5        0.         1.         0.
 1.         0.         0.         0.         0.         0.27212121]
[0.0858396  0.2        0.         0.         1.         0.
 0.         0.         0.         0.33333333 0.         0.12121212]
[0.17516012 0.6        0.         0.33333333 0.         0.
 0.         0.         0.         0.33333333 0.         0.10606061]
[0.49039265 0.4        0.5        0.33333333 1.         0.
 1.         0.         1.         0.33333333 0.         0.44848485]
[0.77332219 0.4        0.         0.         1.         0.
 0.         0.         0.         0.         0.         0.15151515]
[0.37064884 0.4        0.         0.         1.         0.
 1.         0.         0.         0.66666667 1.         0.35757576]]
```

[9]:
```
X_train = df_threeBtrain.values[:,0:11]
Y_train = df_threeBtrain.values[:,11]
len(X_train), len(Y_train)
```

[9]:
```
(436, 436)
```

[10]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[0.12419939 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         1.         0.
```

```
 0.         0.         0.         0.         0.        ]
[0.08382066 0.2        0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[0.12280702 0.2        0.         0.         1.         0.
 0.         0.         0.         0.         0.        ]
[0.55861877 0.4        0.         0.         1.         0.
 0.         0.         0.         0.         0.        ]]
Y = [0.         0.08181818 0.0969697  0.07272727 0.23939394]
```

[11]:
```
# Convert to 2D array (381x11)
m = len(X_train)
X_1 = X_train.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.12419939 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.15065441 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.08382066 0.2        0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.12280702 0.2        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]
 [0.55861877 0.4        0.         0.         1.         0.
  0.         0.         0.         0.         0.        ]]
```

[12]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[12]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[13]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[13]:
```
array([[1.        , 0.12419939, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [1.        , 0.15065441, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [1.        , 0.08382066, 0.2       , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
```

```
       [1.        , 0.12280702, 0.2       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ],
       [1.        , 0.55861877, 0.4       , 0.        , 0.        ,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ]])
```

[14]:
```
theta = np.zeros((12,1))
theta
```

[14]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[15]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m
lambda_value: Regularization parameter.
theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta, lambda_value):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    regularization = lambda_value * np.sum(np.square(theta))
    J = (1 / (2 * m)) * (np.sum(sqrErrors) + regularization)
    return J
```

[16]:
```
lambda_value = 10
cost = compute_loss(X_train, Y_train, theta, lambda_value)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 20.934727519081726
```

[17]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.
lambda_value: Regularization parameter.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations, lambda_value):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        # Use np.multiply() to multiple scalar with the array, theta.
        sum_delta = (alpha / m) * X.transpose().dot(errors) + np.
 ↪multiply(theta,((lambda_value * alpha) / m));
        theta = theta - sum_delta; #theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta, lambda_value)
    return theta, loss_history
```

[18]:
```
theta = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
lambda_value = 10
iterations = 1500
alpha = 0.003
```

[19]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations, lambda_value)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [0.07250306 0.0501732  0.05031337 0.06423585 0.06502243
```

```
0.0726465
 0.03808205 0.03585613 0.02009795 0.07870064 0.0546766  0.05897476]
loss_history = [0.04741442 0.04682257 0.04623973 … 0.00676845 0.00676782
0.00676718]
```

[20]:
```
df_threeBtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	\
239	4000	3	1	2	1	0	0	
113	9620	3	1	1	1	0	1	
325	3460	4	1	2	1	0	0	
66	13200	2	1	1	1	0	1	
479	3660	4	1	2	0	0	0	

	hotwaterheating	airconditioning	parking	prefarea	price
239	0	0	1	0	4585000
113	0	0	2	1	6083000
325	0	1	0	0	4007500
66	1	0	1	0	6930000
479	0	0	0	0	2940000

[21]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = MinMaxScaler()
df_threeBtest[num_vars] = scaler.fit_transform(df_threeBtest[num_vars])
df_threeBtest.head()
```

[21]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	\
239	0.203463	0.50	0.0	0.333333	1.0	0.0	0.0	
113	0.690043	0.50	0.0	0.000000	1.0	0.0	1.0	
325	0.156710	0.75	0.0	0.333333	1.0	0.0	0.0	
66	1.000000	0.25	0.0	0.000000	1.0	0.0	1.0	
479	0.174026	0.75	0.0	0.333333	0.0	0.0	0.0	

	hotwaterheating	airconditioning	parking	prefarea	price
239	0.0	0.0	0.333333	0.0	0.270000

```
113              0.0              0.0  0.666667       1.0  0.412667
325              0.0              1.0  0.000000       0.0  0.215000
66               1.0              0.0  0.333333       0.0  0.493333
479              0.0              0.0  0.000000       0.0  0.113333
```

[22]:
```
dataset_test = df_threeBtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[0.2034632  0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.33333333 0.         0.27      ]
 [0.69004329 0.5        0.         0.         1.         0.
  1.         0.         0.         0.66666667 1.         0.41266667]
 [0.15670996 0.75       0.         0.33333333 1.         0.
  0.         0.         1.         0.         0.         0.215     ]
 [1.         0.25       0.         0.         1.         0.
  1.         1.         0.         0.33333333 0.         0.49333333]
 [0.17402597 0.75       0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.         0.11333333]
 [0.40692641 0.5        0.33333333 0.66666667 1.         1.
  0.         0.         1.         0.         0.         0.42333333]
 [0.19047619 0.5        0.         0.         1.         0.
  0.         0.         0.         0.66666667 0.         0.17      ]
 [0.15844156 0.5        0.         0.33333333 0.         0.
  0.         0.         0.         0.33333333 0.         0.11333333]
 [0.16121212 0.25       0.         0.         1.         0.
  0.         0.         0.         0.33333333 1.         0.16666667]
 [0.63636364 0.75       0.33333333 1.         1.         0.
  0.         0.         1.         0.66666667 0.         0.59333333]
 [0.37662338 0.75       0.33333333 1.         1.         0.
  0.         0.         1.         0.         0.         0.47666667]
 [0.2        0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.         0.         0.21333333]
 [0.15584416 0.5        0.         0.33333333 1.         0.
  1.         0.         0.         0.         0.         0.13333333]
 [0.38181818 0.5        0.         0.         1.         1.
  1.         0.         0.         0.         0.         0.14666667]
 [0.37532468 0.5        0.         0.         1.         0.
  1.         0.         0.         0.         0.         0.22666667]
 [0.06753247 0.5        0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.08666667]
 [0.28138528 0.25       0.         0.33333333 1.         0.
  1.         0.         0.         0.         0.         0.25333333]
 [0.37835498 0.5        0.         0.         1.         0.
  0.         0.         0.         0.         0.         0.18666667]
 [0.12554113 0.5        0.         0.33333333 0.         0.
  1.         0.         0.         0.         0.         0.15333333]
 [0.24675325 0.25       0.         0.         1.         0.
  0.         0.         1.         0.66666667 0.         0.24      ]]
```

[23]:
```
X_test = df_threeBtest.values[:,0:11]
Y_test = df_threeBtest.values[:,11]
len(X_test), len(Y_test)
```

[23]:
```
(109, 109)
```

[24]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[0.2034632  0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.33333333 0.        ]
 [0.69004329 0.5        0.         0.         1.         0.
  1.         0.         0.         0.66666667 1.        ]
 [0.15670996 0.75       0.         0.33333333 1.         0.
  0.         0.         1.         0.         0.        ]
 [1.         0.25       0.         0.         1.         0.
  1.         1.         0.         0.33333333 0.        ]
 [0.17402597 0.75       0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.        ]]
Y = [0.27       0.41266667 0.215      0.49333333 0.11333333]
```

[25]:
```
# Convert to 2D array (164x11)
m = len(X_test)
X_1 = X_test.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[0.2034632  0.5        0.         0.33333333 1.         0.
  0.         0.         0.         0.33333333 0.        ]
 [0.69004329 0.5        0.         0.         1.         0.
  1.         0.         0.         0.66666667 1.        ]
 [0.15670996 0.75       0.         0.33333333 1.         0.
  0.         0.         1.         0.         0.        ]
 [1.         0.25       0.         0.         1.         0.
  1.         1.         0.         0.33333333 0.        ]
 [0.17402597 0.75       0.         0.33333333 0.         0.
  0.         0.         0.         0.         0.        ]]
```

[26]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[26]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 109)
```

[27]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[27]:
```
array([[1.        , 0.2034632 , 0.5       , 0.        , 0.33333333,
        1.        , 0.        , 0.        , 0.        , 0.        ,
        0.33333333, 0.        ],
       [1.        , 0.69004329, 0.5       , 0.        , 0.        ,
        1.        , 0.        , 1.        , 0.        , 0.        ,
        0.66666667, 1.        ],
       [1.        , 0.15670996, 0.75      , 0.        , 0.33333333,
        1.        , 0.        , 0.        , 0.        , 1.        ,
        0.        , 0.        ],
       [1.        , 1.        , 0.25      , 0.        , 0.        ,
        1.        , 0.        , 1.        , 1.        , 0.        ,
        0.33333333, 0.        ],
       [1.        , 0.17402597, 0.75      , 0.        , 0.33333333,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        ]])
```

[28]:
```
theta_test = np.zeros((12,1))
theta_test
```

[28]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[29]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
```

```
J : Loss
"""

def compute_loss_noreg(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[30]:
```
cost_test = compute_loss_noreg(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 5.79399238888889
```

[31]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent_noreg(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss_noreg(X, Y, theta)
    return theta, loss_history
```

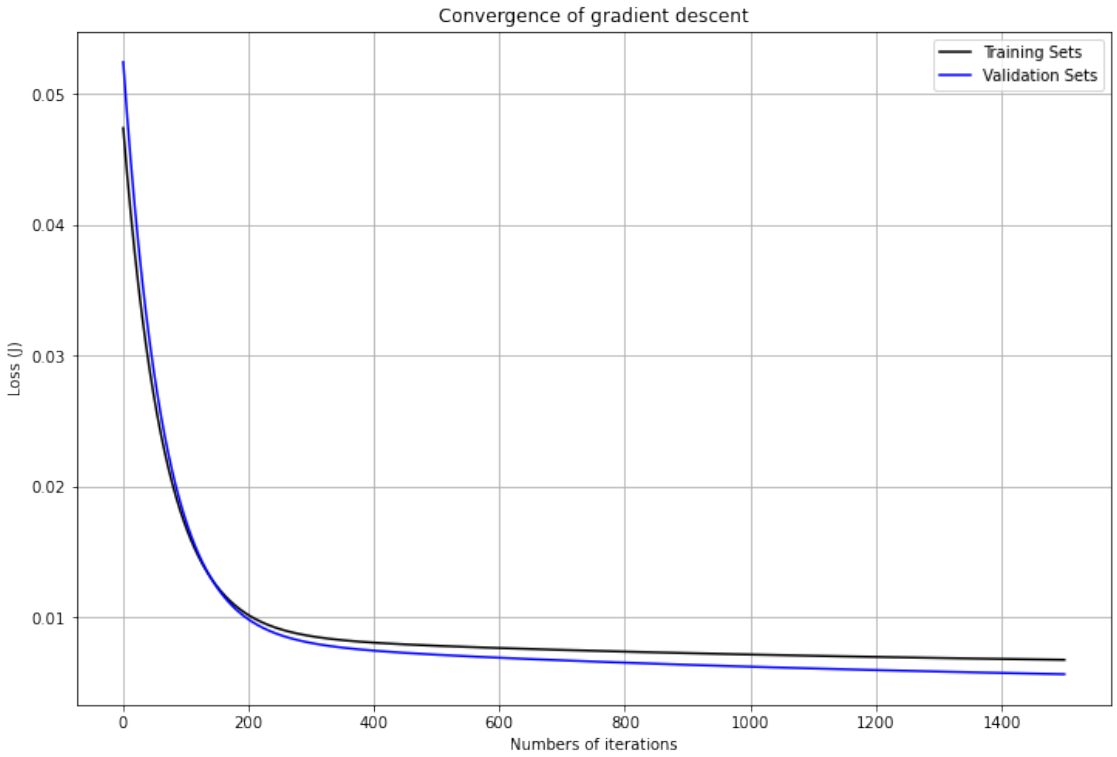
[32]:
```
theta_test = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[33]:
```
theta_test, loss_history_test = gradient_descent_noreg(X_test, Y_test,␣
 ↪theta_test, alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [0.08068021 0.07227611 0.06077027 0.06066932 0.07529487
0.08690136
 0.0128184  0.02678734 0.00876003 0.0921445  0.06943986 0.01541427]
loss_history = [0.05245705 0.05176907 0.05109178 … 0.00566679 0.00566585
0.0056649 ]
```

[35]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[35]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]:

Problem_3b_Standardization

September 28, 2022

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
housing = pd.DataFrame(pd.read_csv('./Housing.csv'))
housing.head()
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

[3]:
```
# Any dataset that has columns with values as 'Yes' or 'No', strings' values␣
 ↪cannot be used.
# However, we can convert them to numerical values as binary.

varlist = ['mainroad', 'guestroom', 'basement', 'hotwaterheating',␣
 ↪'airconditioning', 'prefarea']

def binary_map(x):
    return x.map({'yes': 1, 'no': 0})

housing[varlist] = housing[varlist].apply(binary_map)

housing.head()
```

[3]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	furnished
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	furnished
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	semi-furnished
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	furnished
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	furnished

[4]:
```
# Splitting the Data into Training and Testing Sets
from sklearn.model_selection import train_test_split

# Random seed to randomize the dataset.
np.random.seed(0)
df_train, df_test = train_test_split(housing, train_size = 0.8, test_size = 0.2)
df_train.shape
```

[4]:
```
(436, 13)
```

[5]:
```
df_test.shape
```

[5]:
```
(109, 13)
```

[6]:
```
num_vars = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',␣
 ↪'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'parking',␣
 ↪'prefarea', 'price']
df_threeBtrain = df_train[num_vars]
df_threeBtest = df_test[num_vars]
df_threeBtrain.head()
```

[6]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	price
542	3620	2	1	1	1	0	0	0	0	0	0	1750000
496	4000	2	1	1	1	0	0	0	0	0	0	2660000
484	3040	2	1	1	0	0	0	0	0	0	0	2800000
507	3600	2	1	1	1	0	0	0	0	0	0	2450000

```
252  9860         3          1        1         1          0         0

     hotwaterheating  airconditioning  parking  prefarea    price
542                0                0        0         0  1750000
496                0                0        0         0  2695000
484                0                0        0         0  2870000
507                0                0        0         0  2590000
252                0                0        0         0  4515000
```

[7]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_threeBtrain[num_vars] = scaler.fit_transform(df_threeBtrain[num_vars])
df_threeBtrain.head()
```

[7]:
```
         area  bedrooms  bathrooms   stories  mainroad  guestroom  basement  \
542 -0.716772 -1.294376  -0.573307 -0.933142  0.395599  -0.463125 -0.698609
496 -0.538936 -1.294376  -0.573307 -0.933142  0.395599  -0.463125 -0.698609
484 -0.988206 -1.294376  -0.573307 -0.933142 -2.527811  -0.463125 -0.698609
507 -0.726132 -1.294376  -0.573307 -0.933142  0.395599  -0.463125 -0.698609
252  2.203478  0.052516  -0.573307 -0.933142  0.395599  -0.463125 -0.698609

     hotwaterheating  airconditioning   parking  prefarea     price
542        -0.201427        -0.691351 -0.819149 -0.570288 -1.586001
496        -0.201427        -0.691351 -0.819149 -0.570288 -1.090971
484        -0.201427        -0.691351 -0.819149 -0.570288 -0.999299
507        -0.201427        -0.691351 -0.819149 -0.570288 -1.145974
252        -0.201427        -0.691351 -0.819149 -0.570288 -0.137579
```

[8]:
```
dataset_train = df_threeBtrain.values[:,:]
print(dataset_train[:20,:])
```

```
[[-0.71677205 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -1.5860012 ]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -1.09097091]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
```

```
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.99929863]
[-0.72613182 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -1.14597428]
[ 2.20347795  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.13757923]
[-0.55391195  0.05251643 -0.57330726  0.21291401 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.1925826 ]
[-0.61381451  0.05251643 -0.57330726  0.21291401  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093  0.32555914  1.75350117 -0.10091032]
[ 2.17539862  1.39940847  1.4755613   0.21291401  0.39559913  2.1592447
 -0.69860905 -0.20142689 -0.69135093  1.47026706 -0.57028761  0.24744433]
[-0.70741227 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.63260953]
[-0.76357092 -1.29437561  1.4755613  -0.93314164  0.39559913 -0.46312491
  1.43141575 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.59594061]
[ 3.76656047  0.05251643 -0.57330726  0.21291401  0.39559913 -0.46312491
  1.43141575 -0.20142689  1.44644342  1.47026706  1.75350117  2.63092353]
[-1.14732172 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.96262972]
[-0.39853968  4.09319255  1.4755613   0.21291401  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761  0.68380437]
[-0.30494192 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.79761962]
[-0.07328747  1.39940847  1.4755613  -0.93314164  0.39559913 -0.46312491
  1.43141575 -0.20142689 -0.69135093 -0.81914879 -0.57028761  0.06043289]
[-0.97463386 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093  0.32555914 -0.57028761 -0.85262299]
[-0.37420426  1.39940847 -0.57330726  0.21291401 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093  0.32555914 -0.57028761 -0.94429527]
[ 1.74484894  0.05251643  1.4755613   0.21291401  0.39559913 -0.46312491
  1.43141575 -0.20142689  1.44644342  0.32555914 -0.57028761  1.12749819]
[ 3.64675535  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761 -0.66927844]
[ 0.93990824  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
  1.43141575 -0.20142689 -0.69135093  1.47026706  1.75350117  0.57746453]]
```

[9]:
```
X_train = df_threeBtrain.values[:,0:11]
Y_train = df_threeBtrain.values[:,11]
len(X_train), len(Y_train)
```

[9]:
```
(436, 436)
```

[10]:
```
print('X =', X_train[:5])
print('Y =', Y_train[:5])
```

```
X = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
```

```
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
[-0.98820554 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
[-0.72613182 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
[ 2.20347795  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
 -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]]
Y = [-1.5860012  -1.09097091 -0.99929863 -1.14597428 -0.13757923]
```

[11]:
```
# Convert to 2D array (381x11)
m = len(X_train)
X_1 = X_train.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.71677205 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.53893631 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.98820554 -1.29437561 -0.57330726 -0.93314164 -2.52781141 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [-0.72613182 -1.29437561 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]
 [ 2.20347795  0.05251643 -0.57330726 -0.93314164  0.39559913 -0.46312491
  -0.69860905 -0.20142689 -0.69135093 -0.81914879 -0.57028761]]
```

[12]:
```
m = len(X_train)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[12]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
 436)
```

[13]:
```
X_train = np.hstack((X_0, X_1))
X_train[:5]
```

[13]:
```
array([[ 1.        , -0.71677205, -1.29437561, -0.57330726, -0.93314164,
         0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
       [ 1.        , -0.53893631, -1.29437561, -0.57330726, -0.93314164,
         0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
       [ 1.        , -0.98820554, -1.29437561, -0.57330726, -0.93314164,
        -2.52781141, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
        -0.81914879, -0.57028761],
```

```
        [ 1.        , -0.72613182, -1.29437561, -0.57330726, -0.93314164,
          0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
         -0.81914879, -0.57028761],
        [ 1.        ,  2.20347795,  0.05251643, -0.57330726, -0.93314164,
          0.39559913, -0.46312491, -0.69860905, -0.20142689, -0.69135093,
         -0.81914879, -0.57028761]])
```

[14]:
```
theta = np.zeros((12,1))
theta
```

[14]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[15]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m
lambda_value: Regularization parameter.
theta : 2D array of fitting parameters. Dimension: n,1

Output Parameters
J : Loss
"""

def compute_loss(X, Y, theta, lambda_value):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    regularization = lambda_value * np.sum(np.square(theta))
    J = (1 / (2 * m)) * (np.sum(sqrErrors) + regularization)
    return J
```

[16]:
```
lambda_value = 10
cost = compute_loss(X_train, Y_train, theta, lambda_value)
print("Cost loss for all given theta =", cost)
```

```
Cost loss for all given theta = 218.00000000000006
```

[17]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.
lambda_value: Regularization parameter.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent(X, Y, theta, alpha, iterations, lambda_value):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        # Use np.multiply() to multiple scalar with the array, theta.
        sum_delta = (alpha / m) * X.transpose().dot(errors) + np.
 ↪multiply(theta,((lambda_value * alpha) / m));
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss(X, Y, theta, lambda_value)
    return theta, loss_history
```

[18]:
```
theta = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
lambda_value = 10
iterations = 1500
alpha = 0.003
```

[19]:
```
theta, loss_history = gradient_descent(X_train, Y_train, theta, alpha,␣
 ↪iterations, lambda_value)
print("Final value of theta =", theta)
print("loss_history =", loss_history)
```

```
Final value of theta = [2.49792717e-16 2.75398081e-01 6.97110015e-02
```

```
2.53270759e-01
 1.91044636e-01 8.90255698e-02 9.09053947e-02 8.15391905e-02
 1.19692914e-01 2.15774657e-01 1.16542754e-01 1.59823523e-01]
loss_history = [0.49533858 0.49074438 0.48621641 … 0.16782166 0.16782146
0.16782125]
```

[20]:
```
df_threeBtest.head()
```

[20]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	\
239	4000	3	1	2	1	0	0	
113	9620	3	1	1	1	0	1	
325	3460	4	1	2	1	0	0	
66	13200	2	1	1	1	0	1	
479	3660	4	1	2	0	0	0	

	hotwaterheating	airconditioning	parking	prefarea	price
239	0	0	1	0	4585000
113	0	0	2	1	6083000
325	0	1	0	0	4007500
66	1	0	1	0	6930000
479	0	0	0	0	2940000

[21]:
```
# Many columns contains small integer values excluding areas. Needs to rescale␣
 ↪the variables.
# Advised to use starndarization or normalization, so the coefficients is␣
 ↪comparable.

# Two ways of rescaling:
# 1.) Min-Max Scaling
# 2.) Standardization (For this code.)

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
df_threeBtest[num_vars] = scaler.fit_transform(df_threeBtest[num_vars])
df_threeBtest.head()
```

[21]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	\
239	-0.500735	0.025607	-0.563545	0.272416	0.444750	-0.474045	-0.887066	
113	1.954229	0.025607	-0.563545	-0.915317	0.444750	-0.474045	1.127312	
325	-0.736621	1.421209	-0.563545	0.272416	0.444750	-0.474045	-0.887066	
66	3.518067	-1.369995	-0.563545	-0.915317	0.444750	-0.474045	1.127312	
479	-0.649256	1.421209	-0.563545	0.272416	-2.248456	-0.474045	-0.887066	

	hotwaterheating	airconditioning	parking	prefarea	price

```
239        -0.281439        -0.630425  0.492144 -0.488504 -0.081358
113        -0.281439        -0.630425  1.739673  2.047065  0.801114
325        -0.281439         1.586231 -0.755384 -0.488504 -0.421563
66          3.553168        -0.630425  0.492144 -0.488504  1.300082
479        -0.281439        -0.630425 -0.755384 -0.488504 -1.050428
```

[22]:
```
dataset_test = df_threeBtest.values[:,:]
print(dataset_test[:20,:])
```

```
[[-0.50073521  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421 -0.08135801]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517  1.73967255  2.04706526  0.80111439]
 [-0.73662142  1.42120937 -0.56354451  0.27241586  0.44474959 -0.47404546
   -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421 -0.42156349]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
    1.12731244  3.5531676  -0.63042517  0.49214421 -0.48850421  1.30008243]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -2.24845626 -0.47404546
   -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -1.05042817]
 [ 0.52580664  0.02560738  1.2431129   1.46014902  0.44474959  2.10950231
   -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421  0.86709364]
 [-0.56625916  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
   -0.88706553 -0.28143902 -0.63042517  1.73967255 -0.48850421 -0.69991343]
 [-0.72788489  0.02560738 -0.56354451  0.27241586 -2.24845626 -0.47404546
   -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421 -1.05042817]
 [-0.71390645 -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
   -0.88706553 -0.28143902 -0.63042517  0.49214421  2.04706526 -0.72053194]
 [ 1.68339637  1.42120937  1.2431129   2.64788217  0.44474959 -0.47404546
   -0.88706553 -0.28143902  1.58623108  1.73967255 -0.48850421  1.91863786]
 [ 0.37291742  1.42120937  1.2431129   2.64788217  0.44474959 -0.47404546
   -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421  1.19698986]
 [-0.51820826  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
   -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.43187275]
 [-0.74098968  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.92671708]
 [ 0.399127    0.02560738 -0.56354451 -0.91531729  0.44474959  2.10950231
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.84424303]
 [ 0.36636503  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.34939869]
 [-1.18655253  0.02560738 -0.56354451 -0.91531729 -2.24845626 -0.47404546
   -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -1.21537628]
 [-0.10759152 -1.36999462 -0.56354451  0.27241586  0.44474959 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.18445058]
 [ 0.38165395  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
   -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.59682086]
 [-0.89387889  0.02560738 -0.56354451  0.27241586 -2.24845626 -0.47404546
    1.12731244 -0.28143902 -0.63042517 -0.75538413 -0.48850421 -0.803006  ]
 [-0.28232205 -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
```

```
  -0.88706553 -0.28143902  1.58623108  1.73967255 -0.48850421 -0.26692463]]
```

[23]:
```
X_test = df_threeBtest.values[:,0:11]
Y_test = df_threeBtest.values[:,11]
len(X_test), len(Y_test)
```

[23]:
```
(109, 109)
```

[24]:
```
print('X =', X_test[:5])
print('Y =', Y_test[:5])
```

```
X = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244 -0.28143902 -0.63042517  1.73967255  2.04706526]
 [-0.73662142  1.42120937 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244  3.5531676  -0.63042517  0.49214421 -0.48850421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421]]
Y = [-0.08135801  0.80111439 -0.42156349  1.30008243 -1.05042817]
```

[25]:
```
# Convert to 2D array (164x11)
m = len(X_test)
X_1 = X_test.reshape(m,11)
print("X_1 =", X_1[:5,:])
```

```
X_1 = [[-0.50073521  0.02560738 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902 -0.63042517  0.49214421 -0.48850421]
 [ 1.95422869  0.02560738 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244 -0.28143902 -0.63042517  1.73967255  2.04706526]
 [-0.73662142  1.42120937 -0.56354451  0.27241586  0.44474959 -0.47404546
  -0.88706553 -0.28143902  1.58623108 -0.75538413 -0.48850421]
 [ 3.5180669  -1.36999462 -0.56354451 -0.91531729  0.44474959 -0.47404546
   1.12731244  3.5531676  -0.63042517  0.49214421 -0.48850421]
 [-0.64925616  1.42120937 -0.56354451  0.27241586 -2.24845626 -0.47404546
  -0.88706553 -0.28143902 -0.63042517 -0.75538413 -0.48850421]]
```

[26]:
```
# Create theta zero.
m = len(X_test)
X_0 = np.ones((m,1))
X_0[:5], len(X_0)
```

[26]:
```
(array([[1.],
        [1.],
        [1.],
        [1.],
        [1.]]),
```

109)

[27]:
```
X_test = np.hstack((X_0, X_1))
X_test[:5]
```

[27]:
```
array([[ 1.        , -0.50073521,  0.02560738, -0.56354451,  0.27241586,
         0.44474959, -0.47404546, -0.88706553, -0.28143902, -0.63042517,
         0.49214421, -0.48850421],
       [ 1.        ,  1.95422869,  0.02560738, -0.56354451, -0.91531729,
         0.44474959, -0.47404546,  1.12731244, -0.28143902, -0.63042517,
         1.73967255,  2.04706526],
       [ 1.        , -0.73662142,  1.42120937, -0.56354451,  0.27241586,
         0.44474959, -0.47404546, -0.88706553, -0.28143902,  1.58623108,
        -0.75538413, -0.48850421],
       [ 1.        ,  3.5180669 , -1.36999462, -0.56354451, -0.91531729,
         0.44474959, -0.47404546,  1.12731244,  3.5531676 , -0.63042517,
         0.49214421, -0.48850421],
       [ 1.        , -0.64925616,  1.42120937, -0.56354451,  0.27241586,
        -2.24845626, -0.47404546, -0.88706553, -0.28143902, -0.63042517,
        -0.75538413, -0.48850421]])
```

[28]:
```
theta_test = np.zeros((12,1))
theta_test
```

[28]:
```
array([[0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[29]:
```
"""
Compute loss for linear regression for one time.

Input Parameters
X : 2D array for training example
    m = number of training examples
    n = number of features
Y : 1D array of label/target values. Dimension: m

theta : 2D array of fitting parameters. Dimension: n,1
```

```
Output Parameters
J : Loss
"""

def compute_loss_noreg(X, Y, theta):
    predictions = X.dot(theta) #prediction = h
    errors = np.subtract(predictions, Y)
    sqrErrors = np.square(errors)
    J = 1 / (2 * m) * np.sum(sqrErrors)
    return J
```

[30]:
```
cost_test = compute_loss_noreg(X_test, Y_test, theta_test)
print("Cost loss for all given theta =", cost_test)
```

```
Cost loss for all given theta = 54.49999999999999
```

[31]:
```
"""
Compute loss for l inear regression for all iterations

Input Parameters
X: 2D array, Dimension: m x n
    m = number of training data point
    n = number of features
Y: 1D array of labels/target value for each training data point. Dimension: m
theta: 2D array of fitting parameters or weights. Dimension: (n,1)
alpha : learning rate
iterations: Number of iterations.

Output Parameters
theta: Final Value. 2D array of fitting parameters or weights. Dimension: n,1
loss_history: Contains value of cost at each iteration. 1D Array. Dimension: m
"""

def gradient_descent_noreg(X, Y, theta, alpha, iterations):
    loss_history = np.zeros(iterations)

    for i in range(iterations):
        predictions = X.dot(theta) # prediction (m,1) = temp
        errors = np.subtract(predictions, Y)
        sum_delta = (alpha / m) * X.transpose().dot(errors);
        theta = theta - sum_delta; # theta (n,1)
        loss_history[i] = compute_loss_noreg(X, Y, theta)
    return theta, loss_history
```

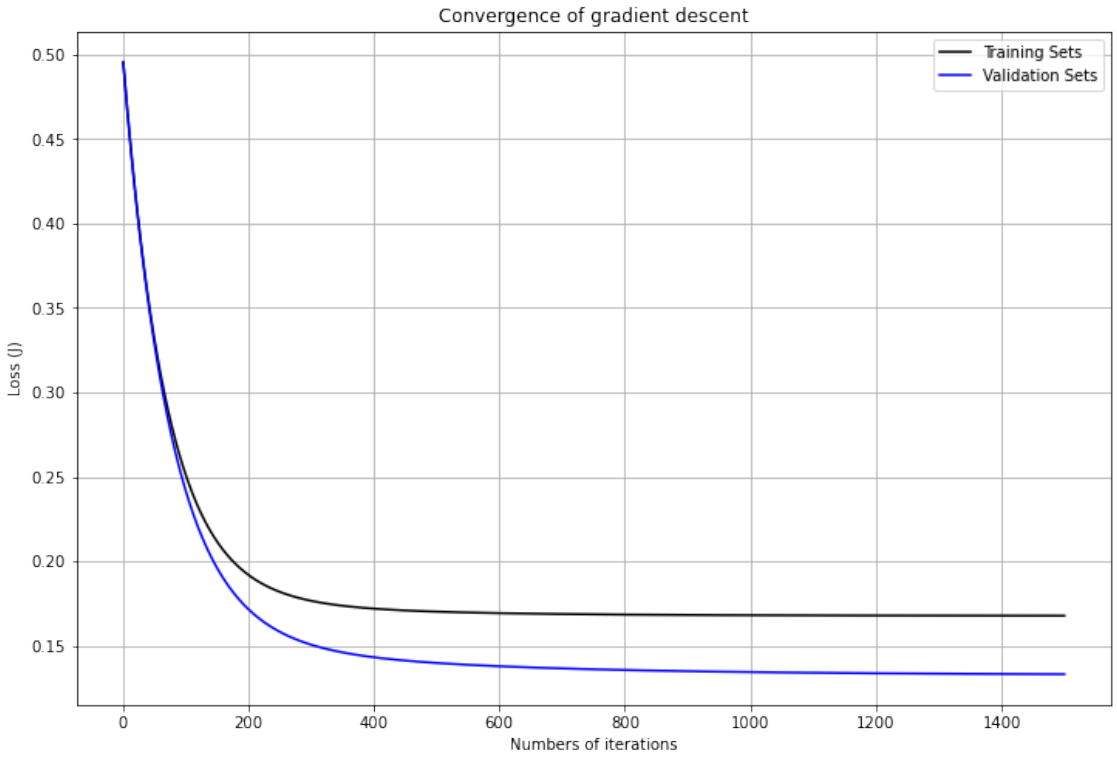
[32]:
```
theta_test = [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]
iterations = 1500
alpha = 0.003
```

[33]:
```
theta_test, loss_history_test = gradient_descent_noreg(X_test, Y_test,␣
  ↪theta_test, alpha, iterations)
print("Final value of theta =", theta_test)
print("loss_history =", loss_history_test)
```

```
Final value of theta = [ 1.27995474e-16  2.70877429e-01  1.98422728e-02
3.05759749e-01
  2.31850131e-01  1.28966576e-01 -4.03787210e-02  1.66329173e-01
  4.35598057e-02  2.61782180e-01  2.34913087e-01  5.10329844e-02]
loss_history = [0.49538399 0.49082909 0.48633448 … 0.13320429 0.13320306
0.13320183]
```

[35]:
```
plt.plot(range(1, iterations + 1), loss_history, color = 'black')
plt.plot(range(1, iterations + 1), loss_history_test, color = 'blue')
plt.rcParams["figure.figsize"] = [12,8]
plt.grid()
plt.legend(['Training Sets', 'Validation Sets'])
plt.xlabel("Numbers of iterations")
plt.ylabel("Loss (J)")
plt.title("Convergence of gradient descent")
```

[35]:
```
Text(0.5, 1.0, 'Convergence of gradient descent')
```



[ ]: