Homework1

September 30, 2021

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

from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import MinMaxScaler, StandardScaler

np.set_printoptions(suppress=True)
```

[2]:
```python
def calculate_cost(X,y,theta):
    """Computes the cost function for linear regression"""

    h = X.dot(theta)
    errors = np.subtract(h,y)
    sqrErrors = np.square(errors)
    J = 1/(2*m) * np.sum(sqrErrors)
    return J

def gradient_descent(X_train,X_test,Y_train,Y_test,theta,alpha,iterations):
    """Computes the gradient descent for linear regression"""

    cost_history_train = np.zeros(iterations)
    cost_history_test = np.zeros(iterations)

    for i in range(iterations):
        h = X_train.dot(theta)
        errors = np.subtract(h,Y_train)
        sum_delta = (alpha/m) * X_train.transpose().dot(errors);
        theta = theta - sum_delta;
        cost_history_train[i] = calculate_cost(X_train,Y_train,theta)
        cost_history_test[i] = calculate_cost(X_test,Y_test,theta)

    return cost_history_train, cost_history_test, theta
```

[3]:
```python
# Problem 1.a
# Reading data from csv

housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
↪["price","area","bedrooms","bathrooms","stories","parking"]))
housing = housing.to_numpy()

# Spliting the training and validation set
np.random.seed(0)

train, test = train_test_split(housing, train_size = 0.7, test_size = 0.3,␣
↪random_state = 42)

m = len(train) # Training samples

# Spliting the inputs and output
Y_train = train[:,0]
X_train = train[:,1:]

Y_test = test[:,0]
X_test = test[:,1:]

# Adding X0 to X_train
X0 = np.ones((len(X_train),1))
X_train = np.hstack((X0, X_train))

# Adding X0 to X_test
X0 = np.ones((len(X_test),1))
X_test = np.hstack((X0, X_test))
```

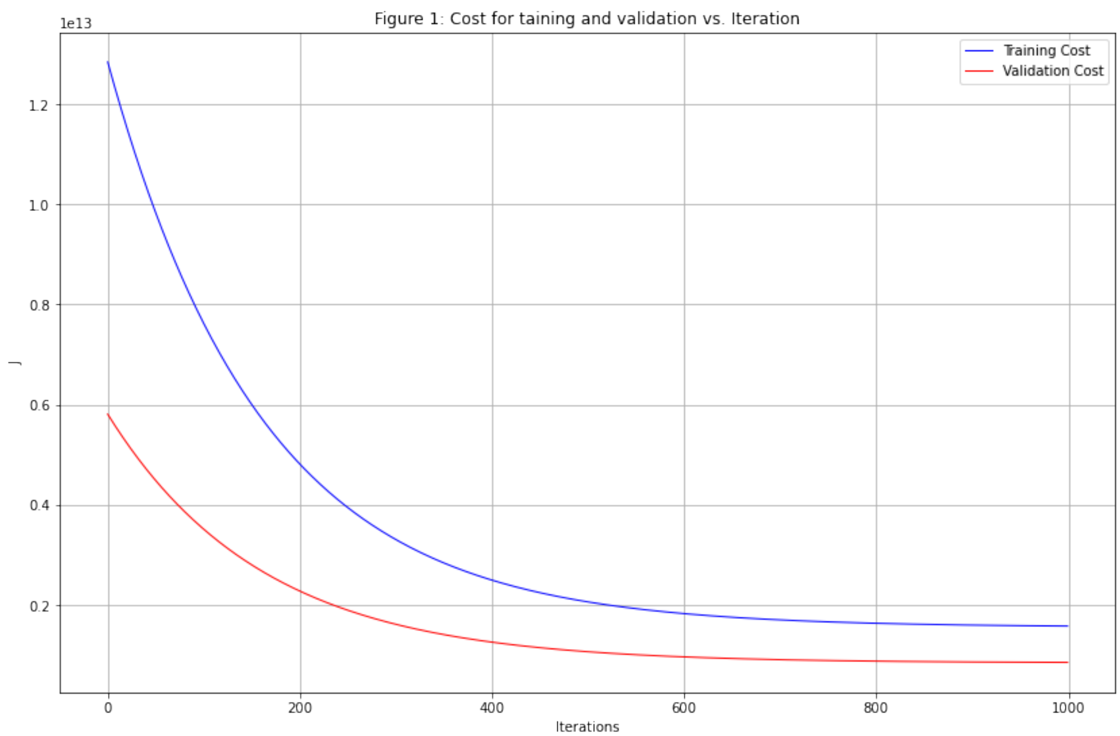
[4]:
```python
# Calculating theta and cost
theta = np.zeros(6)
alpha = 0.0000000001
iterations = 1000

cost_history_train, cost_history_test, theta =␣
↪gradient_descent(X_train,X_test,Y_train,Y_test,theta,alpha,iterations)
print("Theta for X: ", theta)
```

```
Theta for X:  [  0.17254066 817.36415965   0.54640289   0.25046484   0.36083704
   0.13700813]
```

[5]:
```python
# Plotting the Cost vs Iterations
plt.rcParams['figure.figsize'] = [14, 9]
plt.figure()
plt.rcParams['figure.figsize'] = [11, 5]
```

```
plt.plot(cost_history_train[0:len(cost_history_train)], color='blue',␣
 ↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test[0:len(cost_history_test)], color='red', linewidth=1,␣
 ↪label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 1: Cost for taining and validation vs. Iteration")
plt.grid()
plt.legend()
```

[5]: <matplotlib.legend.Legend at 0x7fa9c41e1a60>



[6]:
```
# Problem 1.b
# Reading data from csv

housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
 ↪["price","area","bedrooms","bathrooms","mainroad","guestroom","basement","hotwaterheating",

# Maping the yes/no inputs to 1 and 0
def binary_map(x):
    return x.map({'yes': 1, "no": 0})
```

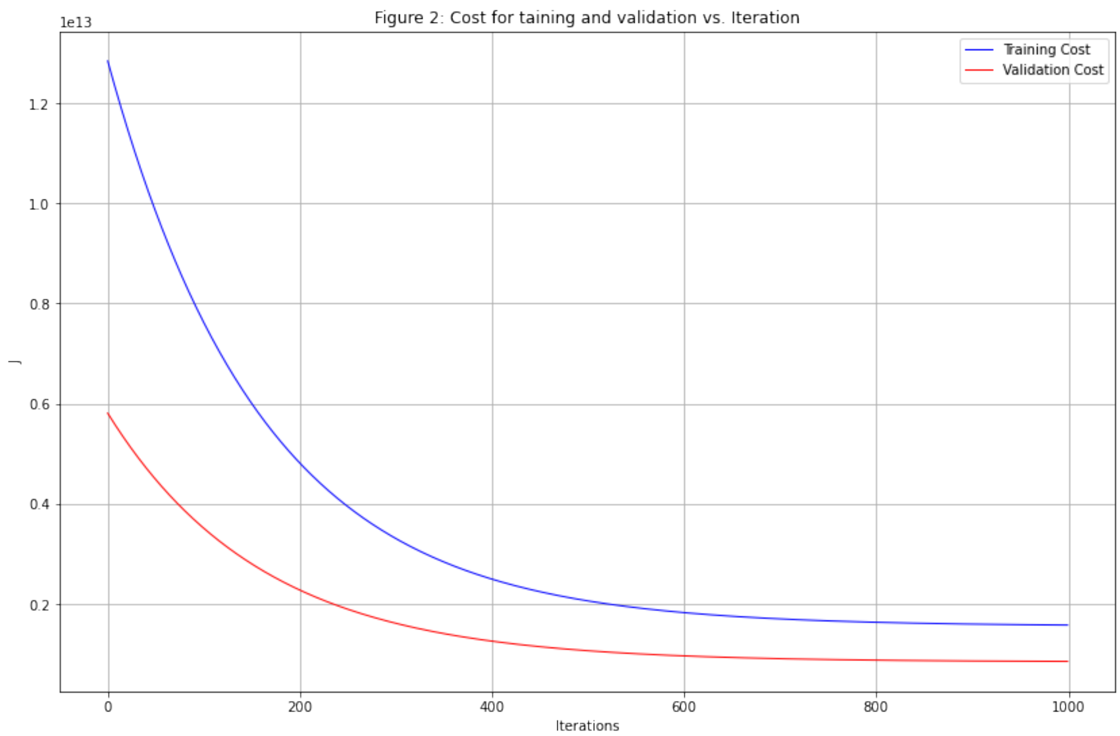
```
binarylist = 
↪["mainroad","guestroom","basement","hotwaterheating","airconditioning","prefarea",]
housing[binarylist] = housing[binarylist].apply(binary_map)

# Converting to numpy
housing = housing.to_numpy()

# Spliting the training and validation set
np.random.seed(0)

train, test = train_test_split(housing, train_size = 0.7, test_size = 0.3, 
↪random_state = 42)

m = len(train) # Training samples

# Spliting the inputs and output
Y_train = train[:,0]
X_train = train[:,1:]

Y_test = test[:,0]
X_test = test[:,1:]

# Adding X0 to X_train
X0 = np.ones((len(X_train),1))
X_train = np.hstack((X0, X_train))

# Adding X0 to X_test
X0 = np.ones((len(X_test),1))
X_test = np.hstack((X0, X_test))
```

[7]:
```
# Calculating theta and cost
theta = np.zeros(12)
alpha = 0.0000000001
iterations = 1000

cost_history_train, cost_history_test, theta = 
↪gradient_descent(X_train,X_test,Y_train,Y_test,theta,alpha,iterations)
```

[8]:
```
# Plotting the Cost vs Iterations
plt.rcParams['figure.figsize'] = [14, 9]
plt.figure()
plt.rcParams['figure.figsize'] = [11, 5]
plt.plot(cost_history_train[0:len(cost_history_train)], color='blue', 
↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test[0:len(cost_history_test)], color='red', linewidth=1, 
↪label="Validation Cost")
plt.xlabel("Iterations")
```

```
plt.ylabel("J")
plt.title("Figure 2: Cost for taining and validation vs. Iteration")
plt.grid()
plt.legend()
```

[8]: <matplotlib.legend.Legend at 0x7fa9c4939340>



[9]:
```
# Problem 2.a
# Reading data from csv
housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
 ↪["price","area","bedrooms","bathrooms","stories","parking"]))

# Spliting the training and validation set
np.random.seed(0)

train, test = train_test_split(housing, train_size = 0.7, test_size = 0.3,␣
 ↪random_state = 42)

m = len(train) # Training samples


# Feature Scaling
# Standardization
```

```
scaler = StandardScaler()
std_train = scaler.fit_transform(train)
std_test = scaler.fit_transform(test)

# Normalization
scaler = MinMaxScaler()
norm_train = scaler.fit_transform(train)
norm_test = scaler.fit_transform(test)

# Spliting the inputs and output for Normalization
Y_train_norm = norm_train[:,0] # Output variable for the training set
X_train_norm = norm_train[:,1:] # Input variable for the training set

Y_test_norm = norm_test[:,0] # Output variable for the validation set
X_test_norm = norm_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_norm),1))
X_train_norm = np.hstack((X0, X_train_norm))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_norm),1))
X_test_norm = np.hstack((X0, X_test_norm))


# Spliting the inputs and output for Standardization
Y_train_std = std_train[:,0] # Output variable for the training set
X_train_std = std_train[:,1:] # Input variable for the training set

Y_test_std = std_test[:,0] # Output variable for the validation set
X_test_std = std_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_std),1))
X_train_std = np.hstack((X0, X_train_std))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_std),1))
X_test_std = np.hstack((X0, X_test_std))
```

[10]:
```
# Calculating theta and cost
theta = np.zeros(6)
alpha = 0.01
iterations = 250

cost_history_train_norm, cost_history_test_norm, theta =␣
 ↪gradient_descent(X_train_norm,X_test_norm,Y_train_norm,Y_test_norm,theta,alpha,iterations)
```

```python
print("Theta for normalized X: ", theta, "\n")

theta = np.zeros(6)
alpha = 0.01
iterations = 200

cost_history_train_std, cost_history_test_std, theta =␣
↪gradient_descent(X_train_std,X_test_std,Y_train_std,Y_test_std,theta,alpha,iterations)
print("Theta for standardized X: ", theta)
```

```
Theta for normalized X:  [0.18999258 0.07446712 0.09050375 0.04504933 0.08173306
0.0739338 ]

Theta for standardized X:  [0.         0.36176458 0.1171842  0.2921215
0.20710648 0.19452094]
```

[22]:
```python
# Plotting the Cost vs Iterations
plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_norm[0:len(cost_history_train_norm)], color='blue',␣
↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_norm[0:len(cost_history_test_norm)], color='red',␣
↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 3: Cost for taining and validation vs. Iterations for␣
↪Normalization")
plt.grid()
plt.legend()

plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_std[0:len(cost_history_train_std)], color='blue',␣
↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_std[0:len(cost_history_test_std)], color='red',␣
↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 4: Cost for taining and validation vs. Iterations for␣
↪Standardization")
plt.grid()
plt.legend()
```

[22]: <matplotlib.legend.Legend at 0x7fa9c00dd040>

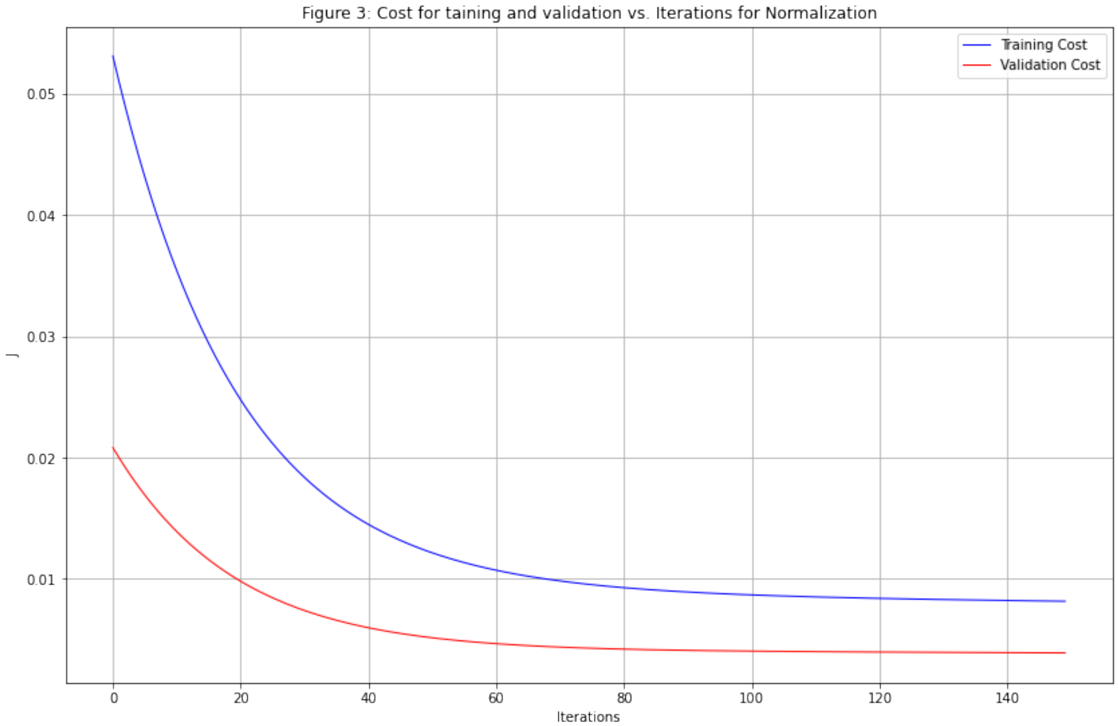


Figure 3: Cost for taining and validation vs. Iterations for Normalization
Training Cost
Validation Cost
0.05
0.04
0.03
0.02
0.01
J
0
20
40
60
80
100
120
140
Iterations

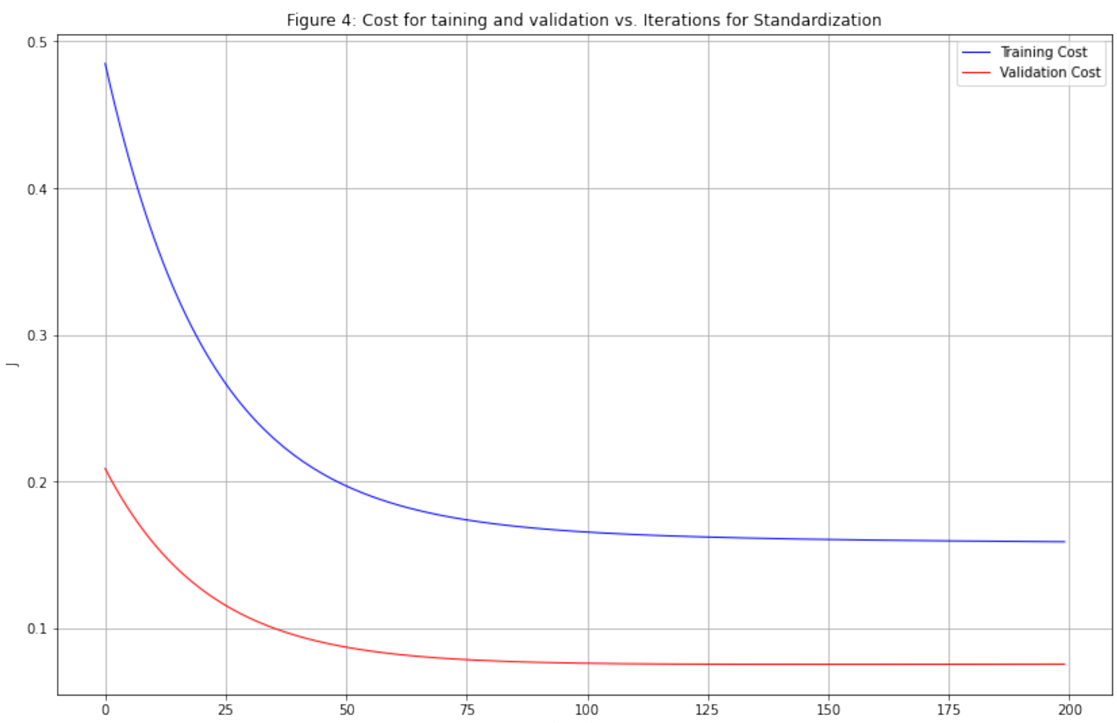


Figure 4: Cost for taining and validation vs. Iterations for Standardization
Training Cost
Validation Cost
0.5
0.4
0.3
0.2
0.1
J
0
25
50
75
100
125
150
175
200
Iterations

[12]:
```
# Problem 2.b

# Reading data from csv

housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
↪["price","area","bedrooms","bathrooms","mainroad","guestroom","basement","hotwaterheating",

# Maping the yes/no inputs to 1 and 0
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

binarylist =␣
↪["mainroad","guestroom","basement","hotwaterheating","airconditioning","prefarea",]
housing[binarylist] = housing[binarylist].apply(binary_map)

# Spliting the training and validation set
np.random.seed(0)

train, test = train_test_split(housing, train_size = 0.7, test_size = 0.3,␣
↪random_state = 42)

m = len(train) # Training samples


# Feature Scaling
# Standardization
scaler = StandardScaler()
std_train = scaler.fit_transform(train)
std_test = scaler.fit_transform(test)

# Normalization
scaler = MinMaxScaler()
norm_train = scaler.fit_transform(train)
norm_test = scaler.fit_transform(test)

# Spliting the inputs and output for Normalization
Y_train_norm = norm_train[:,0] # Output variable for the training set
X_train_norm = norm_train[:,1:] # Input variable for the training set

Y_test_norm = norm_test[:,0] # Output variable for the validation set
X_test_norm = norm_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_norm),1))
X_train_norm = np.hstack((X0, X_train_norm))

# Adding X0 to X_test_norm
```

```
X0 = np.ones((len(X_test_norm),1))
X_test_norm = np.hstack((X0, X_test_norm))


# Spliting the inputs and output for Standardization
Y_train_std = std_train[:,0] # Output variable for the training set
X_train_std = std_train[:,1:] # Input variable for the training set

Y_test_std = std_test[:,0] # Output variable for the validation set
X_test_std = std_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_std),1))
X_train_std = np.hstack((X0, X_train_std))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_std),1))
X_test_std = np.hstack((X0, X_test_std))
```

[13]:

```
# Calculating theta and cost
theta = np.zeros(12)
alpha = 0.01
iterations = 150

cost_history_train_norm, cost_history_test_norm, theta =␣
↪gradient_descent(X_train_norm,X_test_norm,Y_train_norm,Y_test_norm,theta,alpha,iterations)
print("Theta for normalized X: ", theta, "\n")

theta = np.zeros(12)
alpha = 0.01
iterations = 200
cost_history_train_std, cost_history_test_std, theta =␣
↪gradient_descent(X_train_std,X_test_std,Y_train_std,Y_test_std,theta,alpha,iterations)
print("Theta for standardized X: ", theta)
```

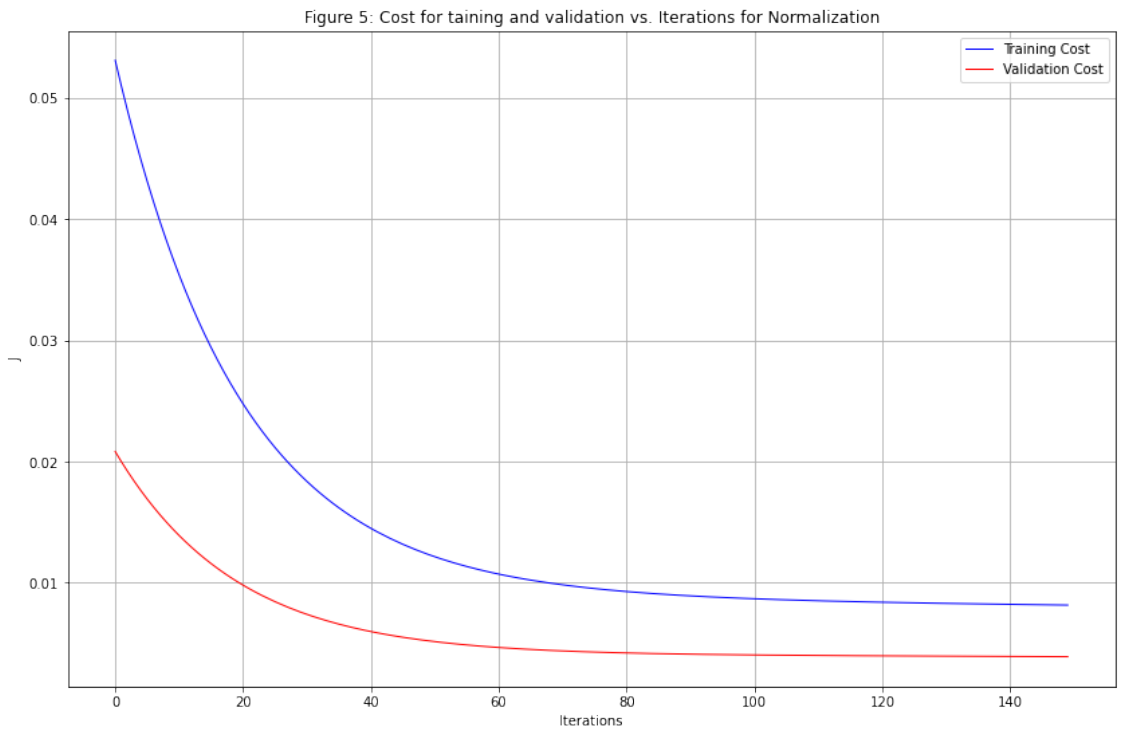
```
Theta for normalized X:  [0.10199588 0.03958284 0.0502271  0.0261717  0.04753288
0.09485726
 0.03096009 0.04488565 0.0103418  0.06258119 0.04103593 0.03786835]

Theta for standardized X:  [0.         0.28136212 0.0922294  0.27227317
0.17908698 0.10572215
 0.08817514 0.11084128 0.07192317 0.18832824 0.15747882 0.12163229]
```

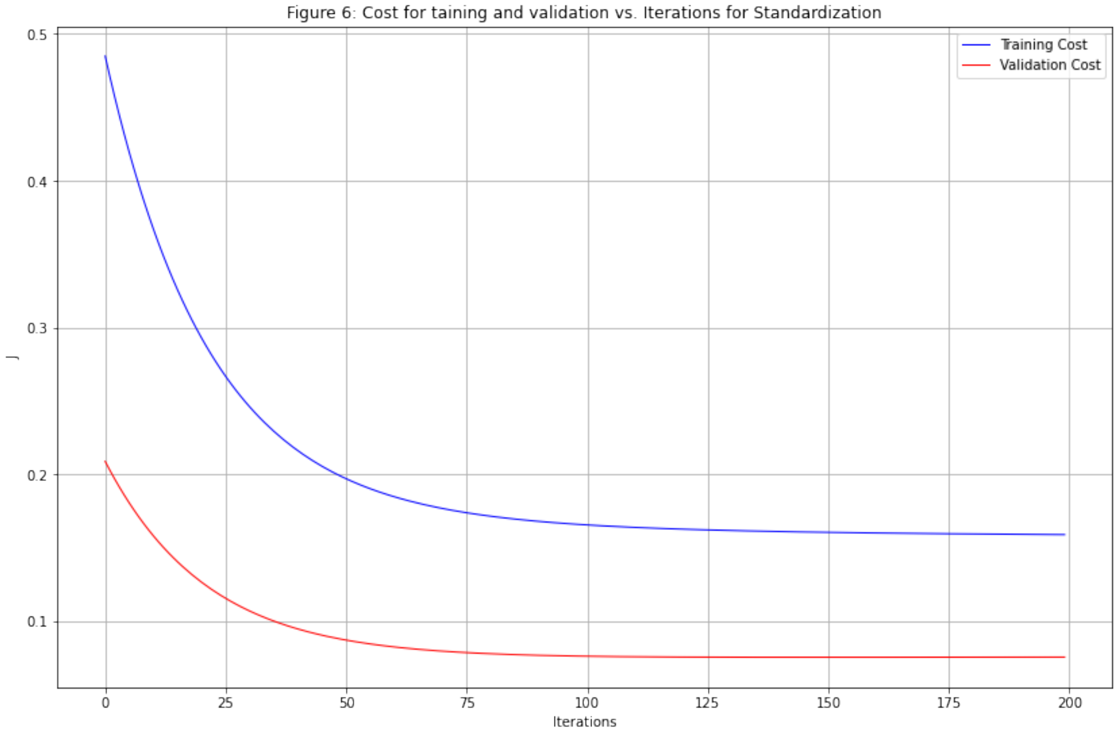
[14]:

```
# Plotting the Cost vs Iterations
plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
```

```
plt.plot(cost_history_train_norm[0:len(cost_history_train_norm)], color='blue',␣
 ↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_norm[0:len(cost_history_test_norm)], color='red',␣
 ↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 5: Cost for taining and validation vs. Iterations for␣
 ↪Normalization")
plt.grid()
plt.legend()

plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_std[0:len(cost_history_train_std)], color='blue',␣
 ↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_std[0:len(cost_history_test_std)], color='red',␣
 ↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 6: Cost for taining and validation vs. Iterations for␣
 ↪Standardization")
plt.grid()
plt.legend()
```

[14]: <matplotlib.legend.Legend at 0x7fa9c523b9a0>





[15]:
```python
# Problem 3

# Cost for regularization on training set
def calculate_cost_training(X,y,theta):
    """Computes the cost function for linear regression with regularization"""

    h = X.dot(theta)
    errors = np.subtract(h,y)
    sqrErrors = np.square(errors)

    reg = np.square(theta[1:]) # Regularization theta 1 to len(theta)
    reg = np.insert(reg, 0, theta[:1], axis=0) # Adding back theta 0

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

# Cost for regularization on validation set
def calculate_cost_test(X,y,theta):
    """Computes the cost function for linear regression"""

    h = X.dot(theta)
```

```
    errors = np.subtract(h,y)
    sqrErrors = np.square(errors)
    J = 1/(2*m) * (np.sum(sqrErrors))
    return J
```

[16]:
```
# Problem 3.a
# Reading data from csv
housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
↪["price","area","bedrooms","bathrooms","stories","parking"]))

# Spliting the training and validation set
np.random.seed(0)

train, test = train_test_split(housing, train_size = 0.7, test_size = 0.3,␣
↪random_state = 42)

m = len(train) # Training samples


# Feature Scaling
# Standardization
scaler = StandardScaler()
std_train = scaler.fit_transform(train)
std_test = scaler.fit_transform(test)

# Normalization
scaler = MinMaxScaler()
norm_train = scaler.fit_transform(train)
norm_test = scaler.fit_transform(test)

# Spliting the inputs and output for Normalization
Y_train_norm = norm_train[:,0] # Output variable for the training set
X_train_norm = norm_train[:,1:] # Input variable for the training set

Y_test_norm = norm_test[:,0] # Output variable for the validation set
X_test_norm = norm_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_norm),1))
X_train_norm = np.hstack((X0, X_train_norm))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_norm),1))
X_test_norm = np.hstack((X0, X_test_norm))


# Spliting the inputs and output for Standardization
```

```
Y_train_std = std_train[:,0] # Output variable for the training set
X_train_std = std_train[:,1:] # Input variable for the training set

Y_test_std = std_test[:,0] # Output variable for the validation set
X_test_std = std_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_std),1))
X_train_std = np.hstack((X0, X_train_std))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_std),1))
X_test_std = np.hstack((X0, X_test_std))
```

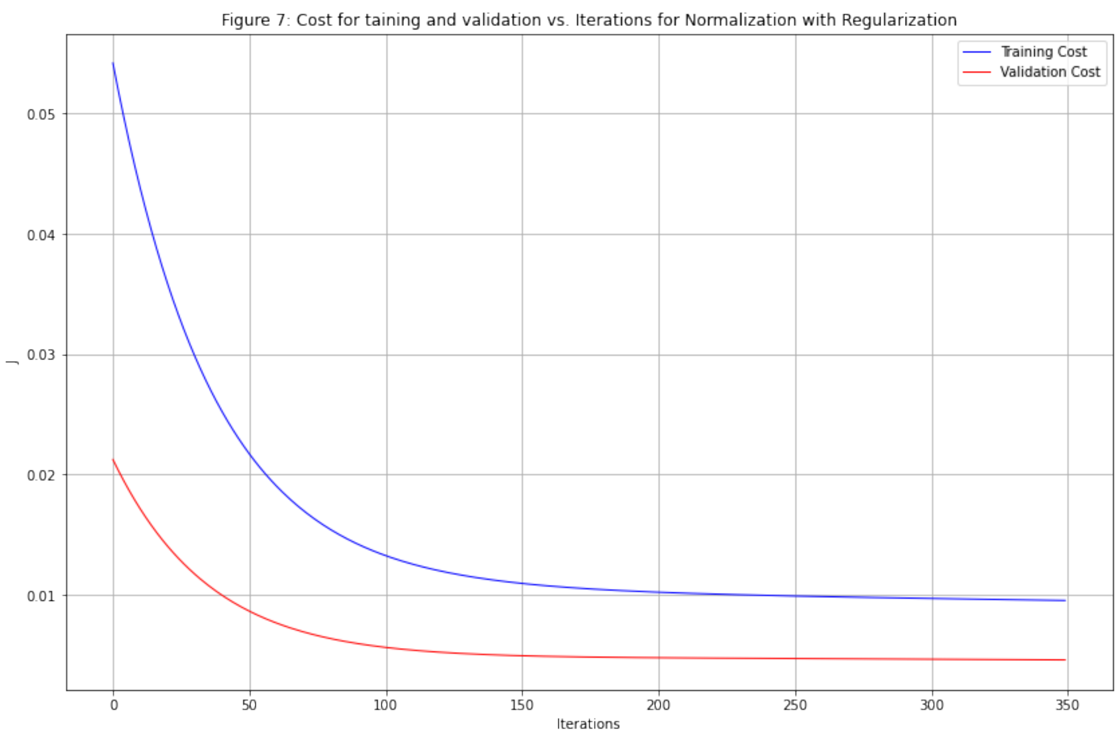
[17]:
```
# Calculating theta and cost
theta = np.zeros(6)
alpha = 0.01
iterations = 350

cost_history_train_norm, cost_history_test_norm, theta =␣
 ↪gradient_descent(X_train_norm,X_test_norm,Y_train_norm,Y_test_norm,theta,alpha,iterations)
print("Theta for normalized with regularization: ", theta, "\n")

theta = np.zeros(6)
alpha = 0.01
iterations = 250

cost_history_train_std, cost_history_test_std, theta =␣
 ↪gradient_descent(X_train_std,X_test_std,Y_train_std,Y_test_std,theta,alpha,iterations)
print("Theta for standardization with regularization", theta)
```

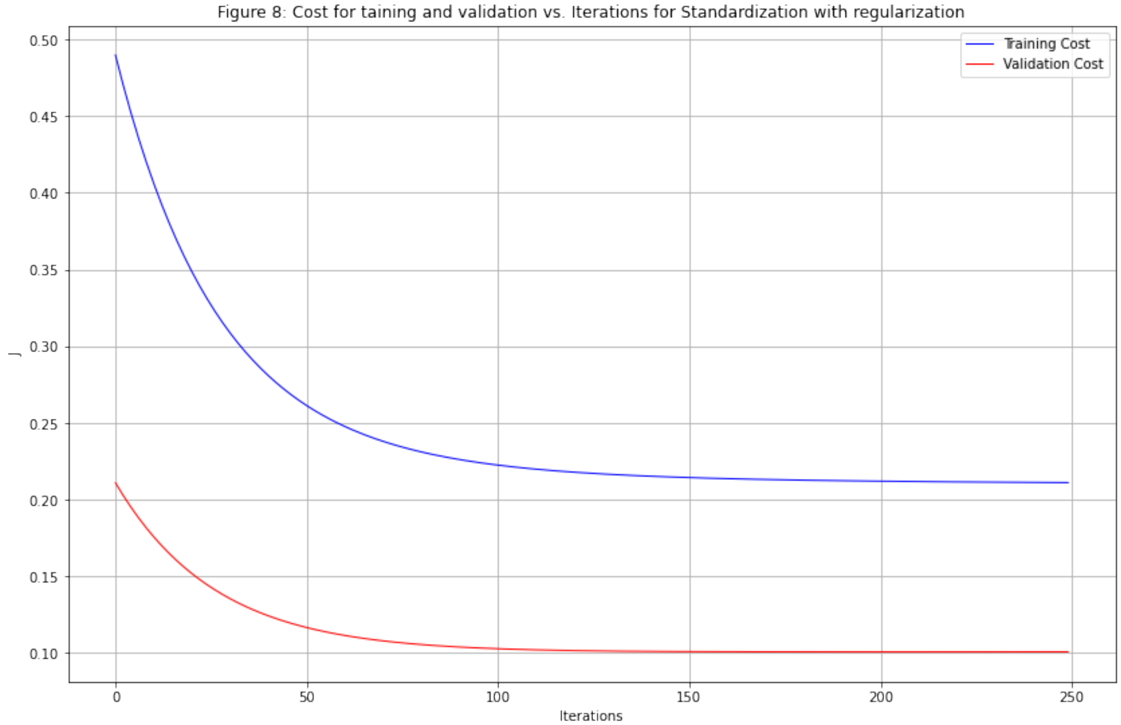
```
Theta for normalized with regularization:  [0.18775205 0.08421515 0.09442677
0.05463201 0.09050374 0.0830125 ]

Theta for standardization with regularization [0.         0.37704584 0.10623171
0.30071024 0.21275141 0.19485301]
```

[18]:
```
# Plotting the Cost vs Iterations
plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_norm[0:len(cost_history_train_norm)], color='blue',␣
 ↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_norm[0:len(cost_history_test_norm)], color='red',␣
 ↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
```

```
plt.title("Figure 7: Cost for taining and validation vs. Iterations for␣
↪Normalization with Regularization")
plt.grid()
plt.legend()

plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_std[0:len(cost_history_train_std)], color='blue',␣
↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_std[0:len(cost_history_test_std)], color='red',␣
↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 8: Cost for taining and validation vs. Iterations for␣
↪Standardization with regularization")
plt.grid()
plt.legend()
```

[18]: <matplotlib.legend.Legend at 0x7fa9c49a83a0>





[19]:
```
# Problem 3.b

# Reading data from csv

housing = pd.DataFrame(pd.read_csv("Housing.csv", usecols =␣
↪["price","area","bedrooms","bathrooms","mainroad","guestroom","basement","hotwaterheating",

# Maping the yes/no inputs to 1 and 0
def binary_map(x):
    return x.map({'yes': 1, "no": 0})

binarylist =␣
↪["mainroad","guestroom","basement","hotwaterheating","airconditioning","prefarea",]
housing[binarylist] = housing[binarylist].apply(binary_map)

# Spliting the training and validation set
np.random.seed(0)

train, test = train_test_split(housing, train_size = 0.7, test_size = 0.3,␣
↪random_state = 42)

m = len(train) # Training samples
```

```
# Feature Scaling
# Standardization
scaler = StandardScaler()
std_train = scaler.fit_transform(train)
std_test = scaler.fit_transform(test)

# Normalization
scaler = MinMaxScaler()
norm_train = scaler.fit_transform(train)
norm_test = scaler.fit_transform(test)

# Spliting the inputs and output for Normalization
Y_train_norm = norm_train[:,0] # Output variable for the training set
X_train_norm = norm_train[:,1:] # Input variable for the training set

Y_test_norm = norm_test[:,0] # Output variable for the validation set
X_test_norm = norm_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_norm),1))
X_train_norm = np.hstack((X0, X_train_norm))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_norm),1))
X_test_norm = np.hstack((X0, X_test_norm))


# Spliting the inputs and output for Standardization
Y_train_std = std_train[:,0] # Output variable for the training set
X_train_std = std_train[:,1:] # Input variable for the training set

Y_test_std = std_test[:,0] # Output variable for the validation set
X_test_std = std_test[:,1:] # Input variable for the training set

# Adding X0 to X_train_norm
X0 = np.ones((len(X_train_std),1))
X_train_std = np.hstack((X0, X_train_std))

# Adding X0 to X_test_norm
X0 = np.ones((len(X_test_std),1))
X_test_std = np.hstack((X0, X_test_std))
```

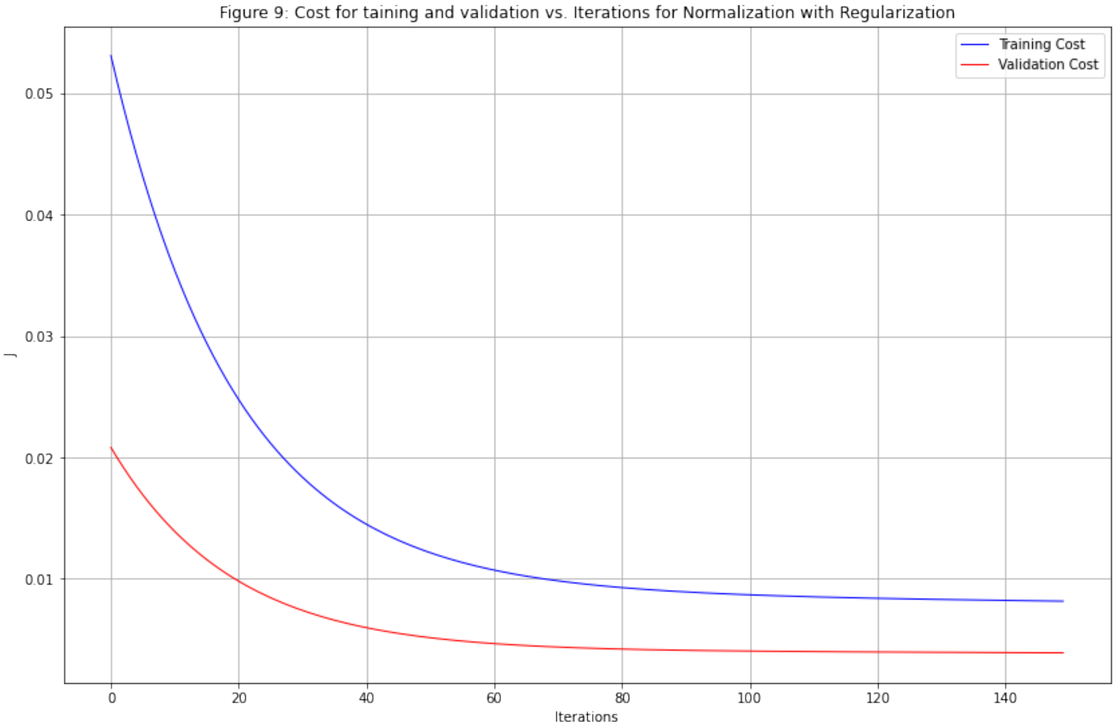
[20]:
```
# Calculating theta and cost
theta = np.zeros(12)
alpha = 0.01
iterations = 150
```

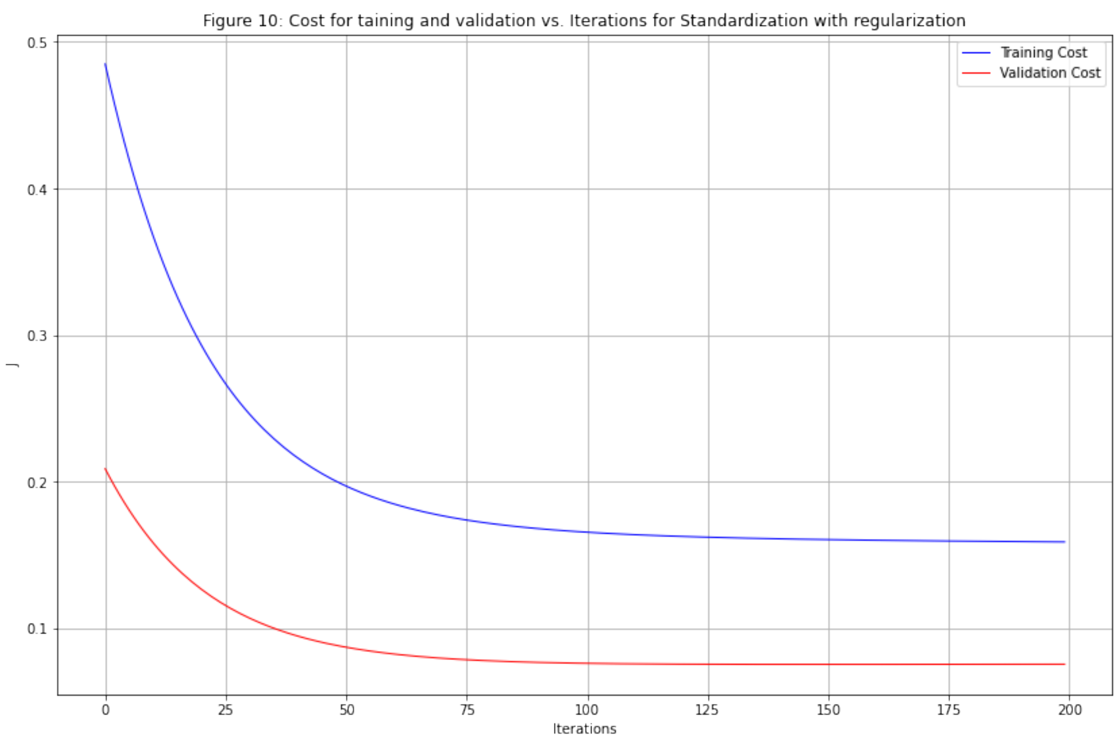
```
cost_history_train_norm, cost_history_test_norm, theta =␣
 ↪gradient_descent(X_train_norm,X_test_norm,Y_train_norm,Y_test_norm,theta,alpha,iterations)
print("Theta for normalized X: ", theta, "\n")

theta = np.zeros(12)
alpha = 0.01
iterations = 200
cost_history_train_std, cost_history_test_std, theta =␣
 ↪gradient_descent(X_train_std,X_test_std,Y_train_std,Y_test_std,theta,alpha,iterations)
print("Theta for standardized X: ", theta)
```

```
Theta for normalized X:  [0.10199588 0.03958284 0.0502271  0.0261717  0.04753288
0.09485726
 0.03096009 0.04488565 0.0103418  0.06258119 0.04103593 0.03786835]

Theta for standardized X:  [0.         0.28136212 0.0922294  0.27227317
0.17908698 0.10572215
 0.08817514 0.11084128 0.07192317 0.18832824 0.15747882 0.12163229]
```

[23]:
```
# Plotting the Cost vs Iterations
plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_norm[0:len(cost_history_train_norm)], color='blue',␣
 ↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_norm[0:len(cost_history_test_norm)], color='red',␣
 ↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 9: Cost for taining and validation vs. Iterations for␣
 ↪Normalization with Regularization")
plt.grid()
plt.legend()

plt.figure()
plt.rcParams['figure.figsize'] = [14, 9]
plt.plot(cost_history_train_std[0:len(cost_history_train_std)], color='blue',␣
 ↪linewidth=1, label="Training Cost")
plt.plot(cost_history_test_std[0:len(cost_history_test_std)], color='red',␣
 ↪linewidth=1, label="Validation Cost")
plt.xlabel("Iterations")
plt.ylabel("J")
plt.title("Figure 10: Cost for taining and validation vs. Iterations for␣
 ↪Standardization with regularization")
plt.grid()
plt.legend()
```

[23]: <matplotlib.legend.Legend at 0x7fa9c5f25550>





```
[ ]:
```