MACHINE LEARNING ASSIGNMENT - 1 [Part -1]

Task 1.

Modify the function calculate_hypothesis.py to return the predicted value for a single specified training example. Include in the report the corresponding lines from your code.

```
def calculate_hypothesis(X, theta, i):
    """
        :param X            : 2D array of our dataset
        :param theta        : 1D array of the trainable parameters
        :param i            : scalar, index of current training sample's row
    """

    hypothesis = 0.0
    ##########################################
    # Write your code here
    # You must calculate the hypothesis for the i-th sample of X, given X, theta
and i.
    hypothesis = X[i, 0] * theta[0] + X[i, 1] * theta[1]

    ##########################################

    return hypothesis
```

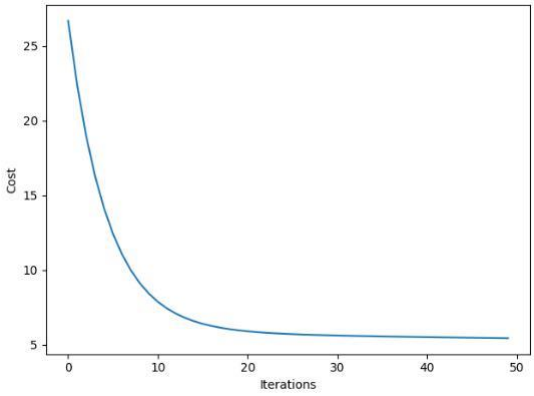
Modified calculate_hypothesis function with gradient descent included. Include the corresponding lines of the code in your report.

```
for i in range(m):
    hypothesis = calculate_hypothesis(X, theta, i)
    ##########################################
    # Write your code here
    # Replace the above line that calculates the hypothesis, with a call to the
"calculate_hypothesis" function

    ##########################################/
    output = y[i]
```

Observe what happens when you use a very high or very low learning rate. Document and comment on your findings in the report.

High learning rate ought to increase the cost very high. At the same time, a very low learning rate like alpha = 0.001 is gives low cost,It is optimum to choose a lesser value of alpha for achieving better learning rate .
With learning rate, 0.023, we can see a very less error rate as illustrated below. .

The minimum cost is 5.42692 for alpha = 0.023



Cost Graph

Task 2:

Modify the functions calculate_hypothesis and gradient_descent to support the new hypothesis function. Your new hypothesis function's code should be sufficiently general so that we can have any number of extra variables. Include the relevant lines of the code in your report.

```
hypothesis = 0
for j in range(len(theta)):
    hypothesis = hypothesis + X[i, j] * theta[j]
#########################################/

return hypothesis
```

gradient_descent.py

```
# Gradient Descent loop
for it in range(iterations):

    # initialize temporary theta, as a copy of the existing theta array
    theta_temp = theta.copy()

    sigma = np.zeros((len(theta)))
    #sigma = 0.0
    for j in range(len(theta)):
        for i in range(m):
            hypothesis = calculate_hypothesis(X, theta, i)
            output = y[i]
            sigma[j] = sigma[j] + (hypothesis - output) * X[i, j]
        theta_temp[j] = theta_temp[j] - (alpha/m) * sigma[j]

    #theta = np.array([theta_0, theta_1, theta_2])
    theta = theta_temp.copy()
```

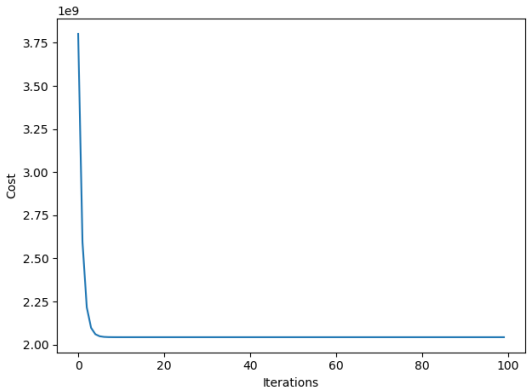
Run ml_assgn1_2.py and see how different values of alpha affect the convergence of the algorithm. Print the theta values found at the end of the optimization. Does anything surprise you? Include the values of theta and your observations in your report.

Dataset normalization complete.
Gradient descent finished.
Minimum cost: 2043280050.60283, on iteration #48
293081.4643348961
472277.8551463628
The values of theta when alpha = 1
340412.6595744681
109447.7964696418
-6578.35485416127

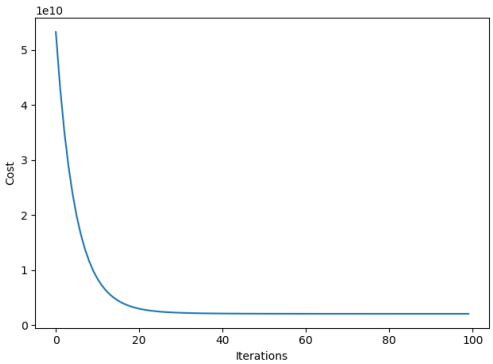
Dataset normalization complete.
Gradient descent finished.
Minimum cost: 2043462824.61817, on iteration #100
293214.1635457116
472159.98841419816
The values of theta when alpha = .1
340403.6177380308
108803.37852265747
-5933.941340198017

Dataset normalization complete.
Gradient descent finished.
Minimum cost: 10596969344.16698, on iteration #100
183865.19798768786
316034.4730065222
The values of theta when alpha = .01
215810.61679137868
61446.187813607605
20070.133137958157

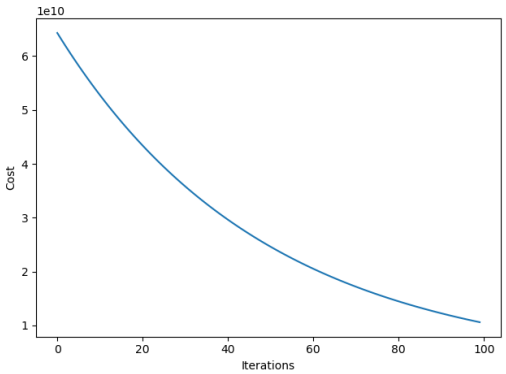
Alpha = 1



Alpha = .1



Alpha = .01



From the illustrations, it is pretty clear that when the value of alpha is decreasing, w_2 is becoming positive (i.e.. increasing) and w_0 and w_1 are decreasing. The value of minimum cost is also increasing as alpha is decreasing.

Add some lines of code in ml_assgn1_2.py to make predictions of house prices. Add the lines of the code that you wrote in your report. Include as well the predictions that you make for the prices of the houses above.

ml_assgn1_2.py

```
# Write your code here
# Create two new samples: (1650, 3) and (3000, 4)
x1 = [1650, 3]
x2 = [3000, 4]
#print(x1)
x1 = (x1 - mean_vec)/std_vec
x2 = (x2 - mean_vec)/std_vec
#print (x1[0,0])
#print(theta_final)
y1 = theta_final[0] + theta_final[1] * x1[0,0] + theta_final[2] * x1[0,1]
y2 = theta_final[0] + theta_final[1] * x2[0,0] + theta_final[2] * x2[0,1]
print(y1)
print(y2)
print("The values of theta when alpha = .01")
print(theta_final[0])
print(theta_final[1])
print(theta_final[2])
```

For 1650 sq. ft. and 3 bedrooms the cost is : 293081.4643348961
For 3000 sq. ft. and 4 bedrooms the cost is : 472277.8551463628

Task 3

Note that the punishment for having more terms is not applied to the bias. This cost function has been implemented already in the function compute_cost_regularised. Modify gradient_descent to use the compute_cost_regularised method instead of your report and a brief explanation

gradient_descent.py

```
iteration_cost = compute_cost_regularised(X, y, theta, l)
```

compute_cost_regularised.py

```
for i in range(m):
    hypothesis = calculate_hypothesis(X, theta, i)
    output = y[i]
    squared_error = (hypothesis - output)**2
    J = J + squared_error
J = J/(2*m)

return J
```

Regularization is not applied to theta[0] (ie.bias term) as seen from the above code where 'i' starts from 1 and not 0 .
Regularization is not necessary for bias term because it doesn't contribute any to the curvature of the model and hence neglected.

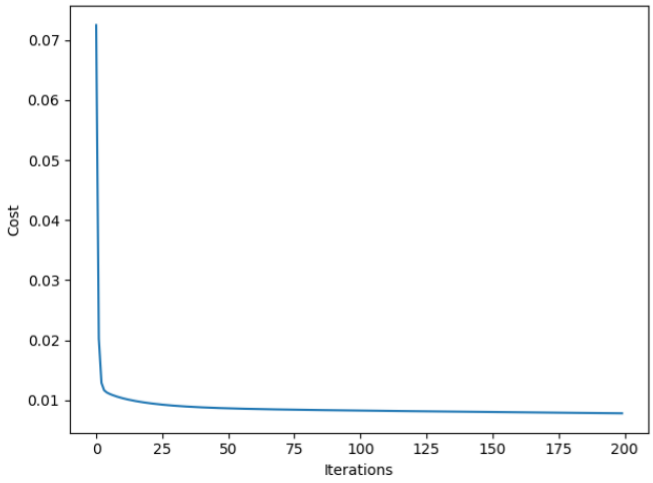
Modify gradient_descent to incorporate the new cost function. Again, we do not want to punish the bias term. Include the relevant lines of your code in your report.

gradient_descent.py

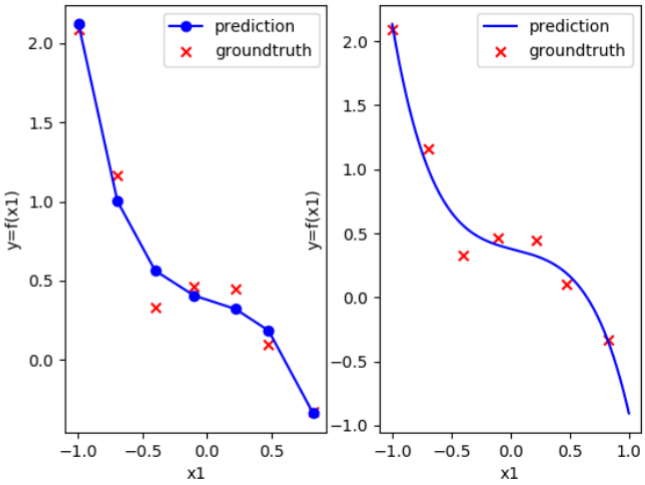
```
theta_temp = theta.copy()
sigma = np.zeros((len(theta)))
for i in range(m):
    hypothesis = calculate_hypothesis(X, theta, i)
    output = y[i]
    sigma[0] = sigma[0] + (hypothesis - output)
theta_temp[0] = theta_temp[0] - (alpha/m) * sigma[0]

for j in range(1,len(theta)):
    for i in range(m):
        hypothesis = calculate_hypothesis(X, theta, i)
        output = y[i]
        sigma[j] = sigma[j] + (hypothesis - output) * X[i, j]
    theta_temp[j] = (theta_temp[j] * (1 - ((alpha *l)/m))) - (alpha/m) * sigma[j]

theta = theta_temp.copy()
iteration_cost = compute_cost_regularised(X, y, theta, l)
cost_vector = np.append(cost_vector, iteration_cost)
```

After gradient_descent has been updated, run ml_assgn1_3.py. This will plot the hypothesis function found at the end of the optimization. First of all, find the best value of alpha to use in order to optimize best. Report the value of alpha that you found in your report.

The best value of alpha is 1 for which the minimum cost is 0.00780. When alpha is increased to 2 or more than 2, the model is worst in predicting values. Decreasing alpha to 0.1 gives 0.00957 minimum cost and alpha = 1 is considered to be the best because of the cost graph that we get.



Cost graph when alpha = 1



-Hypothesis function graph

Next, experiment with different values of λ and see how this affects the shape of the hypothesis. Note that gradient_descent will have to be modified to take an extra parameter, l (which represents λ, used for regularization). Include in your report the plots for a few different values of λ and comment.

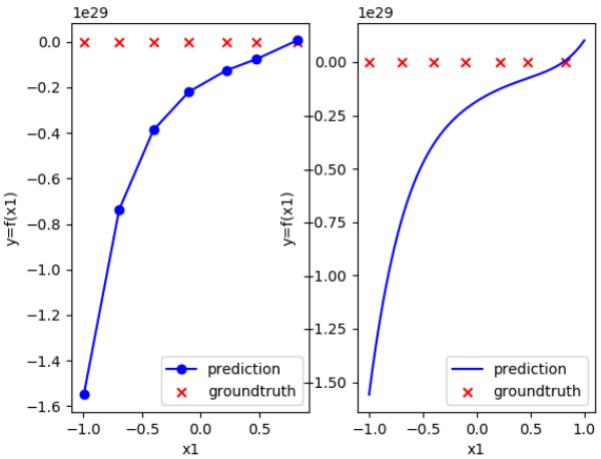


Fig 6:- Hypothesis graph when λ = 10

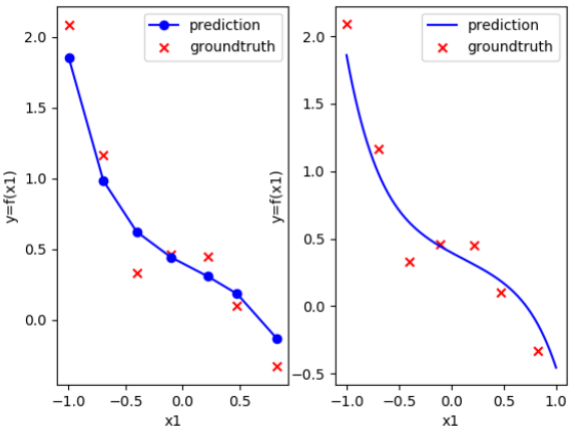
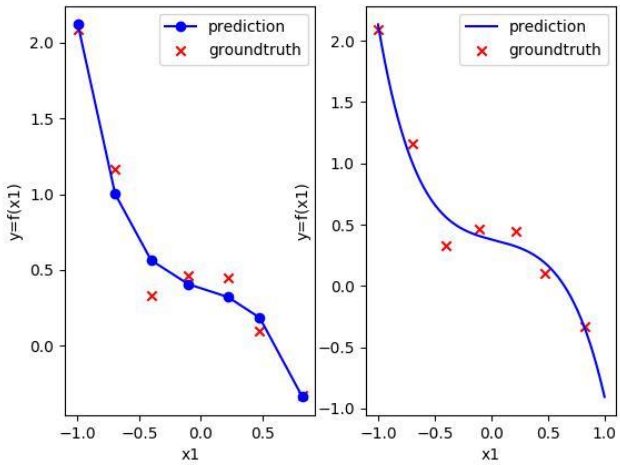


Fig 7:- Hypothesis graph when λ = 1



Hypothesis graph when λ = 0.0001

	Minimum Cost
λ=10	0.57065
λ = 2	0.08338
λ = 1	0.05295
λ =0.1	0.01437
λ = 0.0001	0.00781
λ = 0.00001	0.00780
λ = 0.000001	0.00780

.....

theta_final = gradient_descent(X, y, theta, alpha, iterations, do_plot, l)

.....

Extra parameter has been added and found that as larger value of lambda the cost is very high while compared to lesser value of lambda. It is illustrated clearly for lambda tending to zero, the minimum cost remains constant.