KUSHAGRA GUPTA

Assignment 1 Part 1

1. Linear Regression with One Variable

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.

Answer :

- Added the below code in the following file for calculating hypothesis: **calculate_hypothesis.py**

##CODE##
```
hypothesis = X[i,0] * theta[0] + X[i,1] * theta[1]
```
##CODE##

- Used the specified function in the file **gradient_descent.py** for calculating hypothesis**:**

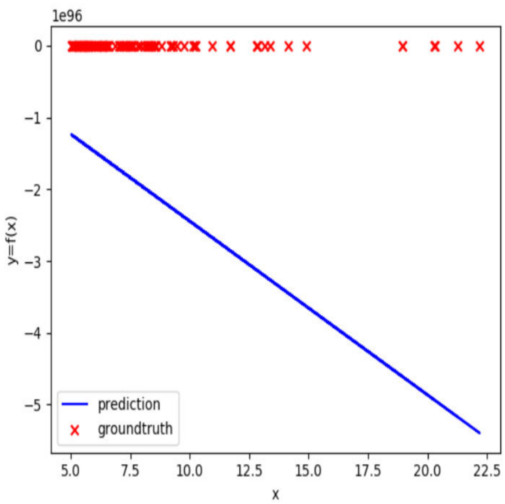
##CODE##
```
# update temporary variable for theta_0
    sigma = 0.0
    for i in range(m):
      #  hypothesis = X[i, 0] * theta[0] + X[i, 1] * theta[1]
        ##########################################
        # Write your code here
        # Replace the above line that calculates the hypothesis, with a call to the "calculate_hypothesis" function
        hypothesis = calculate_hypothesis(X,theta,i)

        ########################################/
        output = y[i]
        sigma = sigma + (hypothesis - output)
    theta_0 = theta_0 - (alpha/m) * sigma

    # update temporary variable for theta_1
    sigma = 0.0
    for i in range(m):

       # hypothesis = X[i,0] * theta[0] + X[i,1] * theta[1]
        ##########################################
        # Write your code here
        # Replace the above line that calculates the hypothesis, with a call to the "calculate_hypothesis" function
        hypothesis = calculate_hypothesis(X,theta,i)
        ########################################/
        output = y[i]
        sigma = sigma + (hypothesis - output) * X[i, 1]
    theta_1 = theta_1 - (alpha/m) * sigma

    # update theta, using the temporary variables
    theta = np.array([theta_0, theta_1])
```

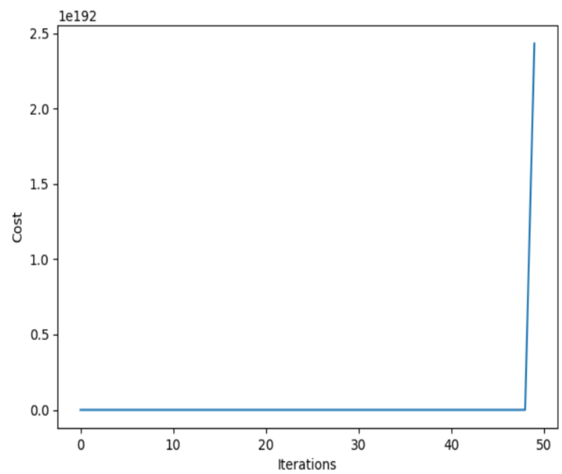
```
# append current iteration's cost to cost_vector
iteration_cost = compute_cost(X, y, theta)
cost_vector = np.append(cost_vector, iteration_cost)
```

##CODE##

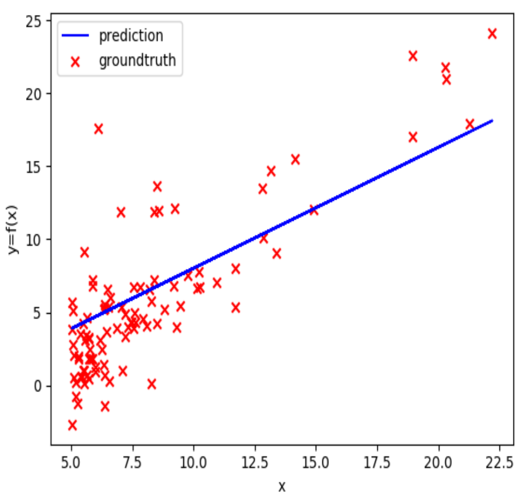
- Observations after keeping different values of learning rate

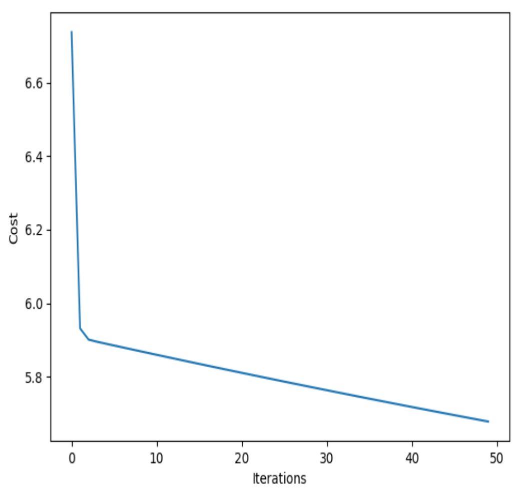
When the learning is kept high
Minimum cost: 172570.09522





When the learning is kept low
Minimum cost: 5.67829





The large learning rate value allows the model to learn faster but the cost is too much as can seen from the plots and cost above, for high learning rate the value of cost boils down to Minimum cost: 172570.09522
which is quite high. When small value is taken the model is learning more optimal set of weights but it takes long time to descent or train the model. Here the cost is Minimum cost: 5.67829 which is significantly high but low as compared to the high learning rate.

2. Linear Regression with Multiple Variables

TASK2 : 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.

ANSWER :

- Added extra line of codes to make **Calculate_hypothesis** work for multi variables.

##CODE##

```
a = X.shape[1]
   for cnt in range(a):
       hypothesis = hypothesis + X[i,cnt] * theta[cnt]
```

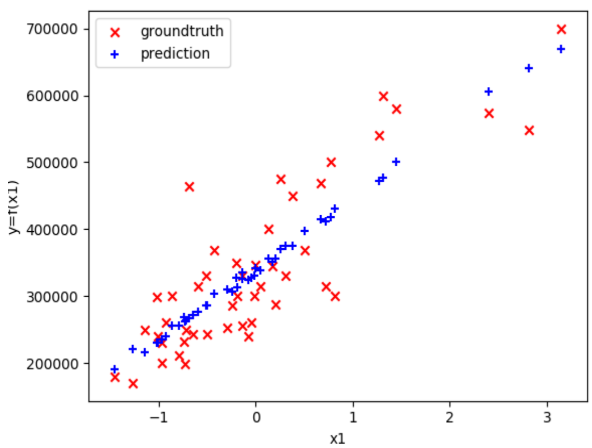
##CODE##

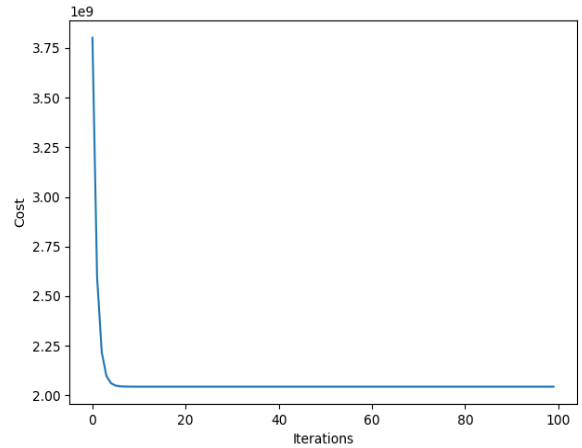
- Changed **gradient_descent** to use the hypothesis function and also work for multi-variables.

##CODE##

```
for i in range(m):
          ############################################
          # Write your code here
          # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
          hypothesis = calculate_hypothesis(X,theta,i)
          ###########################################/
          output = y[i]
          ############################################
          # Write your code here
          # Adapt the code, to compute the values of sigma for all the elements of theta
          for j in range(len(sigma)):
             sigma[j] = sigma[j] + (hypothesis - output) * X[i,j]
          ###########################################/

       # update theta_temp
       ############################################
       # Write your code here
       # Update theta_temp, using the values of sigma
       for i in range(len(sigma)):
          theta_temp[i] = theta_temp[i] - (alpha/m) * sigma[i]
       ###########################################/
       # copy theta_temp to theta
       theta = theta_temp.copy()
```

##CODE##





Explanation : When the value of alpha is kept low , keeping **alpha= 0.01**,value of **Minimum cost: 10596969344.16698** and Theta value are **[215810.61679137868, 61446.187813607605, 20070.133137958157]**
When the value of alpha is kept high, keeping **alpha =2**, value of **Minimum cost: 78551946778.75638**
And Theta value are **[-1.028828040345424e+22, -2.205457950943646e+37, -2.205457950943644e+37]**
When the value of alpha is kept **0.5 or 1 , Minimum cost: 2043280050.60283** and Theta value are **[340412.6595744681,109447.79646870408,-6578.354853223538].**

- Changed code to calculate prediction for a house with 1650 sq. ft. and 3 bedrooms cost

#CODE#

```
# plot predictions for every iteration?
do_plot = True

# call the gradient descent function to obtain the trained parameters theta_final
theta_final = gradient_descent(X_normalized, y, theta, alpha, iterations, do_plot)
print('Theta value is {} {} {}'.format(theta_final[0],theta_final[1],theta_final[2]))

########################################
# Write your code here
# Create two new samples: (1650, 3) and (3000, 4)
# Calculate the hypothesis for each sample, using the trained parameters theta_final
# Make sure to apply the same preprocessing that was applied to the training data
# Print the predicted prices for the two samples
w = theta_final.transpose()
#data = [1,(1604-(mean_vec[0,0]))/(std_vec[0,0]),(3-(mean_vec[0,1]))/(std_vec[0,1])]
#testing values
data = np.array([[1, 1650, 3], [1, 3000, 4]])
#normalising the testing values
def norm_data():
   m=np.array([0,mean_vec[0,0],mean_vec[0,1]])
   st=np.array([1,std_vec[0,0],std_vec[0,1]])
   f_data=[]
   dim=data.shape[0]
```

```
    for k in range(dim):
        nomdata = data[k]-m
        adf = [nomdata/st]
        f_data+= adf
    return f_data
#prediction
prediction = np.dot(norm_data(), w)
print("Predicted value is {}".format(prediction))
```

Predicted House value is [293081.4643349 472277.85514636]

3. Regularized Linear Regression:

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 compute_cost. Include the relevant lines of the code in your report and a brief explanation :

- Changed **calculate_hypothesis** to handle multi variables

##CODE##

```
a = X.shape[1]
    for cnt in range(a):
        hypothesis = hypothesis + X[i,cnt] * theta[cnt]
```

##CODE##

- Code change in **gradient_descent** to handle multi variable

##CODE##

```
# Write your code here
            # Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
            hypothesis = calculate_hypothesis(X,theta,i)
            ########################################/
            output = y[i]
            ########################################
            # Write your code here
            # Adapt the code, to compute the values of sigma for all the elements of theta
            for j in range(len(sigma)):
                sigma[j] = sigma[j] + (hypothesis - output) * X[i,j]
            ########################################/

        # update theta_temp
        ########################################
        # Write your code here
        # Update theta_temp, using the values of sigma
        for i in range(len(sigma)):
            theta_temp[i] = theta_temp[i] - (alpha/m) * sigma[i]
```

```
########################################/

# copy theta_temp to theta
theta = theta_temp.copy()
l=10
# append current iteration's cost to cost_vector
iteration_cost = compute_cost_regularised(X, y, theta,l)
cost_vector = np.append(cost_vector, iteration_cost)
```

##CODE##

- **gradient_descent** has been update with **the cost function:**

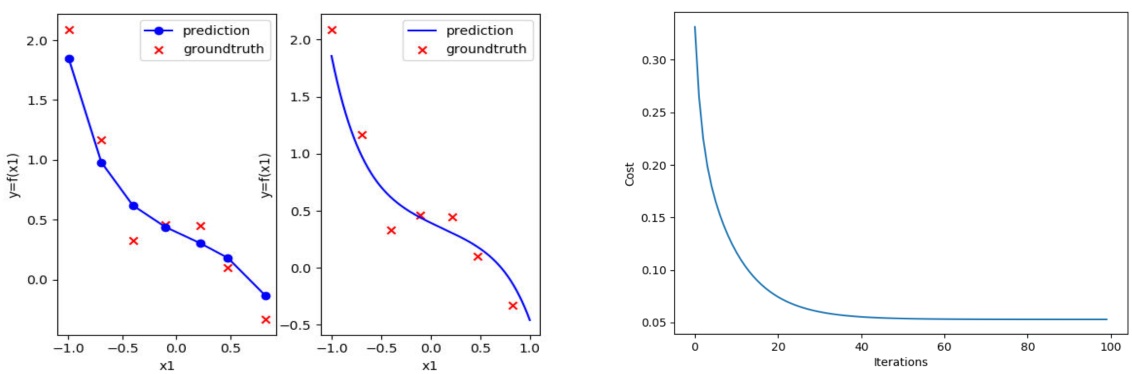
##CODE##

```
# Calculate the hypothesis for the i-th sample of X, with a call to the "calculate_hypothesis" function
    hypothesis = calculate_hypothesis(X,theta,i)
    ########################################/
    output = y[i]
    #########################################
    # Write your code here
    # Adapt the code, to compute the values of sigma for all the elements of theta
    for j in range(len(sigma)):
        sigma[j] = sigma[j] + (hypothesis - output) * X[i,j]
    ########################################/

  # update theta_temp
  #########################################
  # Write your code here
  # Update theta_temp, using the values of sigma
  for i in range(1,len(theta)):
    theta_temp[i] = theta_temp[i] * ((1-(alpha*l)/m))-((alpha/m)*sigma[i])
    theta_temp[0] = theta_temp[0] - (alpha/m) * sigma[0]
  ########################################/

  # copy theta_temp to theta
  theta = theta_temp.copy()

  # append current iteration's cost to cost_vector
  iteration_cost = compute_cost_regularised(X, y, theta,l)
  cost_vector = np.append(cost_vector, iteration_cost)
```

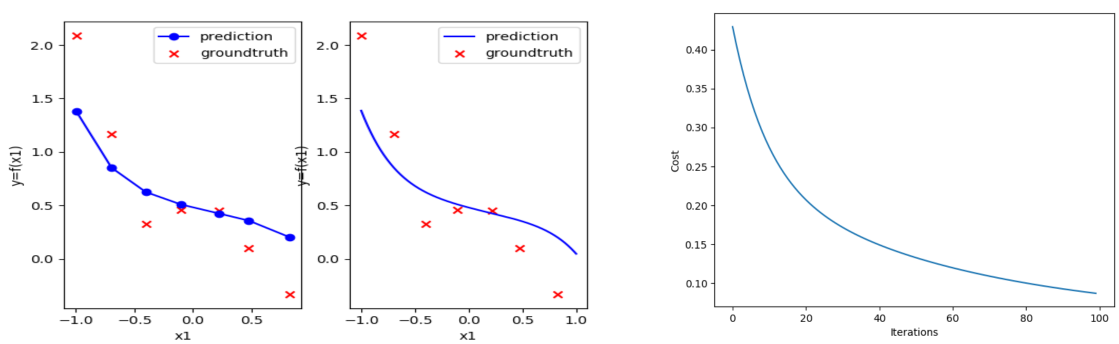
##CODE##

- **Observation for alpha values:**
  When different values of alpha were kept, alpha value of **0.37** was the best value on 100 iterations.
  Below are the given plots for different values of alpha with corresponding cost function. As can been seen that for low and the high values of alpha the cost is more and plot visibly shows that at 0.37 the model has learned well rather than at other values.

Alpha = 0.37
Minimum cost: 0.05295



Alpha = 0.01
Minimum cost: 0.08726



Alpha=2
Minimum cost: 19.33338