ECS708P Machine Learning: Assignment 1 Part 1

Animesh Devendra Chourey

210765551

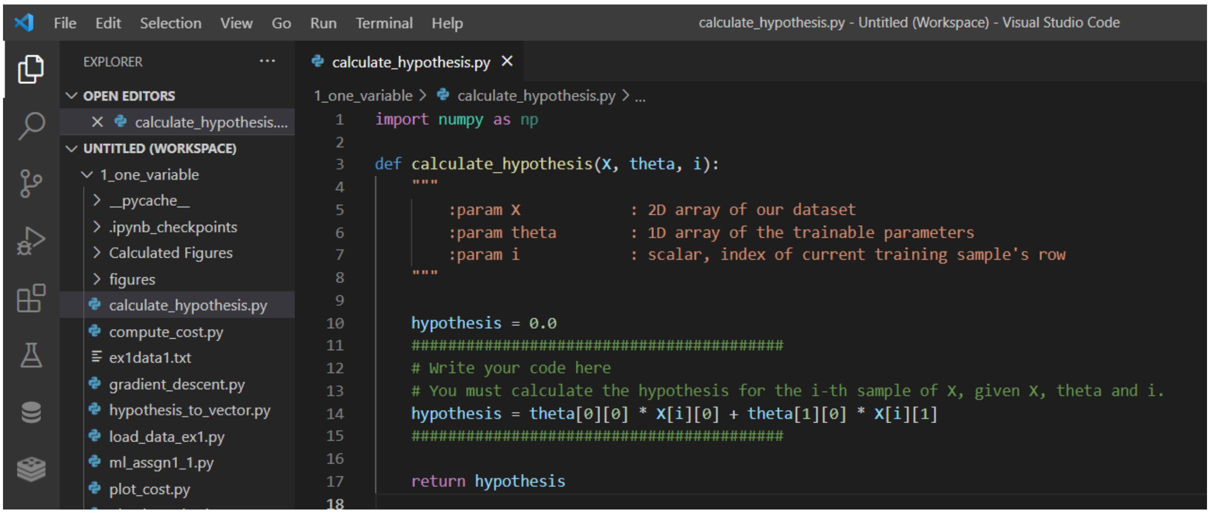
1. Linear Regression

Linear Regression is a supervised learning technique. Here we predict results with continuous output, meaning we are trying to map input variables to some continuous function. We try to fit our dataset with the help of gradient descent algorithm which basically helps us find the best parameters.

1.1 Linear Regression with One Variable

Task 1:

hypothesis = theta[0][0] * X[i][0] + theta[1][0] * X[i][1]



gradient_descent.py will call the calculate_hypothesis()

hypothesis = calculate_hypothesis(X, theta, i)

```
able > gradient_descent.py > gradient_descent

# Gradient Descent
for it in range(iterations):
    # get temporary variables for theta's parameters
    theta_0 = theta[0]
    theta_1 = theta[1]

    # 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] #(This is the line we had to comment)
        #########################################
        # 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
```

Learning Rate:

Learning Rate which is represented by alpha **α.** Alpha determines the size of each step we need to take. Larger value means larger steps and smaller value means smaller steps. Here to find the optimal value of alpha for this dataset the number of iterations is fixed on 100.

Alpha (α)	**Minimum Cost**
1	172570.09522
0.001	5.89503
0.0001	17.36882

- If alpha is too large i.e., alpha = 1

  Gradient Descent might miss the global minimum and it overshoots. It has started diverging as has been the case here.
- If alpha is too small i.e., alpha = 0.0001

  Gradient Descent can become too slow to converge. It is going to need a lot of iterations to converge and will need a lot of steps to reach global minimum.

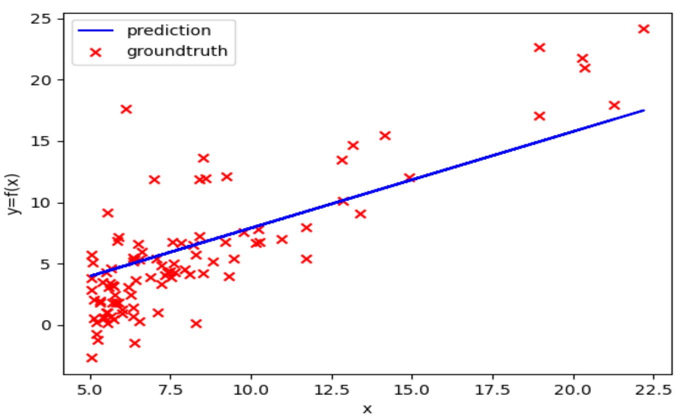


Fig 1. Alpha = 0.001

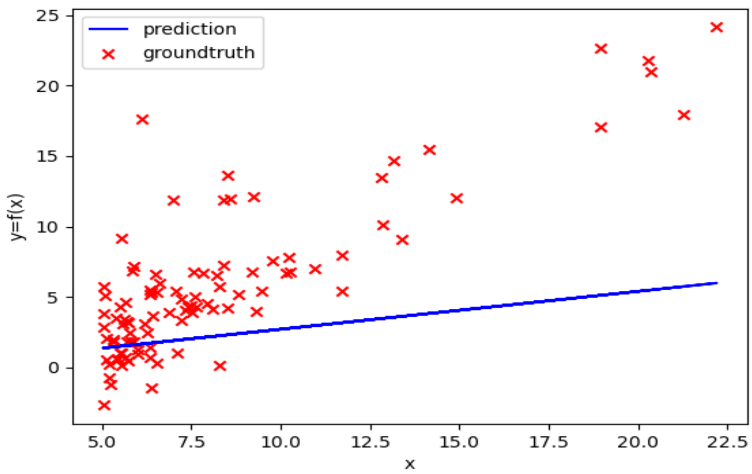


Fig 2. Alpha = 0.0001

1.2 Linear Regression with One Variable

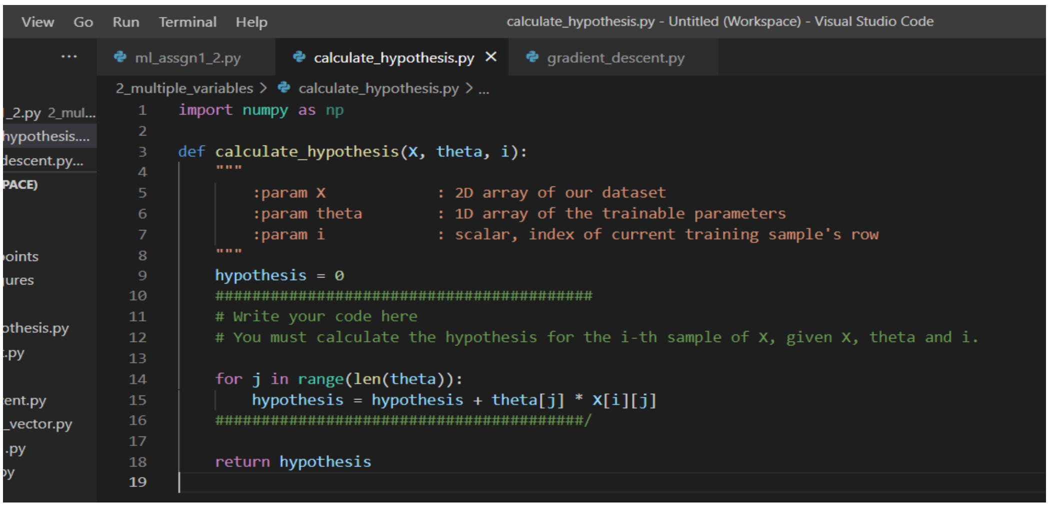
Task 2:

- **calculate_hypothesis.py**

  Hypothesis-

  for j in range(len(theta)):

  hypothesis = hypothesis + theta[j] * X[i][j]




```python
import numpy as np

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
    #########################################
    # Write your code here
    # You must calculate the hypothesis for the i-th sample of X, given X, theta and i.

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

    return hypothesis
```

- **gradient_descent.py**
  - Calling the hypothesis function

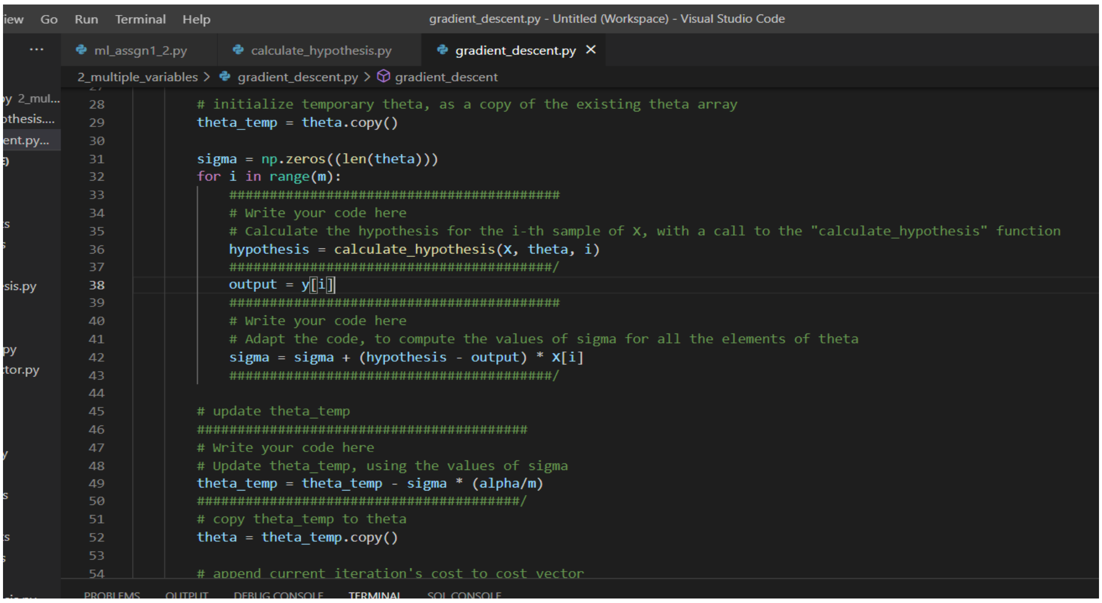
    hypothesis = calculate_hypothesis(X, theta, i)

- Sigma for all the values of theta

  sigma = sigma + (hypothesis - output) * X[i]

- Updating theta_temp

  theta_temp = theta_temp - sigma * (alpha/m)




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

sigma = np.zeros((len(theta)))
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
    sigma = sigma + (hypothesis - output) * X[i]
    ########################################/

# update theta_temp
##########################################
# Write your code here
# Update theta_temp, using the values of sigma
theta_temp = theta_temp - sigma * (alpha/m)
########################################/
# copy theta_temp to theta
theta = theta_temp.copy()

# append current iteration's cost to cost vector
```

Alpha	Minimum Cost	Theta [θ_0, θ_1, θ_2]
0.01	10596969344.16698, on iteration #100	[215810.61679138 61446.18781361 20070.13313796]
0.05	2062616003.82372, on iteration #100	[3.38397236e+05 1.03161481e+05 -3.22620198e+02]
0.001	53852390240.30811	[32409.9584134 9932.44103785 4936.53500492]
0.6	2043280050.60283, on iteration #90	[3.38397236e+05 1.03161481e+05 -3.22620198e+02]
0.4	2043280050.60283, on iteration #100	[340412.65957447 109447.79624289 -6578.35462741]

For alpha 0.4 and 0.6 cost is same but at different iterations. In the case of alpha = 0.6 after reaching minimum, the cost is increasing. Therefore, the optimal value of our algorithm to use for this specific dataset is alpha =0.4.

Prediction of house prices

```
X_test = np.array([[1650, 3],[3000, 4]]) #Created the test array

X_test_normalized = np.zeros(X_test.shape) #stored zero array of same shape as test set

for i in range(len(X_test_normalized)):

        for j in range(len(X_test_normalized)):

                X_test_normalized[i][j] = ( X_test[i][j] - mean_vec[0][j] ) / std_vec[0][j]    #normalized the test set

column_of_ones = np.ones((X_test_normalized.shape[0], 1))

X_test_normalized = np.append(column_of_ones, X_test_normalized, axis=1)

y_predicted = np.zeros(X_test[0].shape)

for i in range(len(X_test)):

        y_predicted[i] = calculate_hypothesis(X_test_normalized, theta_final, i)
```

print("House 1 Predicted Price = ",y_predicted[0])

print("House 2 Predicted Price = ",y_predicted[1])

Run Terminal Help ml_assgn1_2.py - Untitled (Workspace) - Visual Studio Code

ml_assgn1_2.py × calculate_hypothesis.py gradient_descent.py

2_multiple_variables > ml_assgn1_2.py > ...

```
theta_final = gradient_descent(X_normalized, y, theta, alpha, iterations, do_plot)
print("Theta Final ->",theta_final)

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

X_test = np.array([[1650, 3],[3000, 4]]) #Created the test array
X_test_normalized = np.zeros(X_test.shape) #stored zero array of same shape as test set
for i in range(len(X_test_normalized)):
    for j in range(len(X_test_normalized)):
        X_test_normalized[i][j] = ( X_test[i][j] - mean_vec[0][j] ) / std_vec[0][j]    #normalized the test set

column_of_ones = np.ones((X_test_normalized.shape[0], 1))
X_test_normalized = np.append(column_of_ones, X_test_normalized, axis=1)

y_predicted = np.zeros(X_test[0].shape)
for i in range(len(X_test)):
    y_predicted[i] = calculate_hypothesis(X_test_normalized, theta_final, i)

print("House 1 Predicted Price = ",y_predicted[0])
print("House 2 Predicted Price = ",y_predicted[1])
########################################/
```

Output-

Minimum cost: 2043280050.60283, on iteration #100
Theta Final -> [340412.65957447 109447.79624289 -6578.35462741]
House 1 Predicted Price = 293081.46438477084
House 2 Predicted Price = 472277.8551080717

1.3 Regularized Linear Regression

Task 3

gradient_descent.py

- Implementing the cost regularized function

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

- Updating theta along with performing regularization

    theta_temp[0] = theta_temp[0] - (alpha/m) * sigma[0]

    for j in range(1, len(theta_temp)):

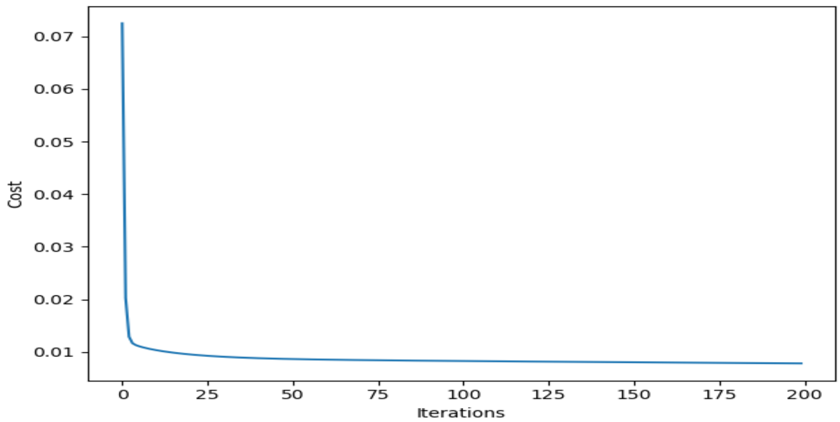
        theta_temp[j] = theta_temp[j] * (1 - (alpha * l)/m) - (alpha/m) * sigma[i]

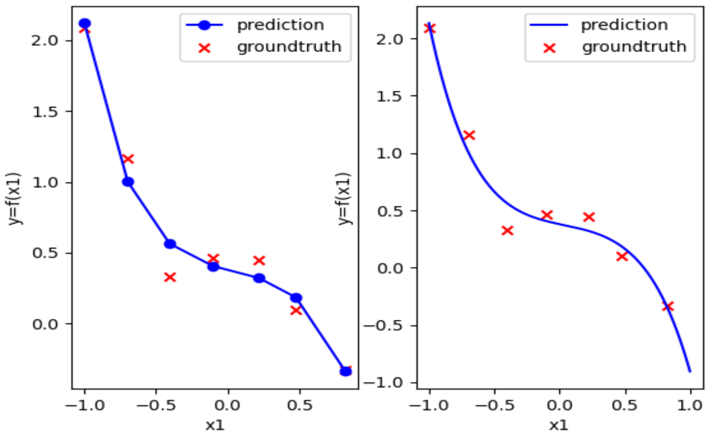
ml_assgn1_3.py

- Best Value of Alpha

    Alpha = 1.0

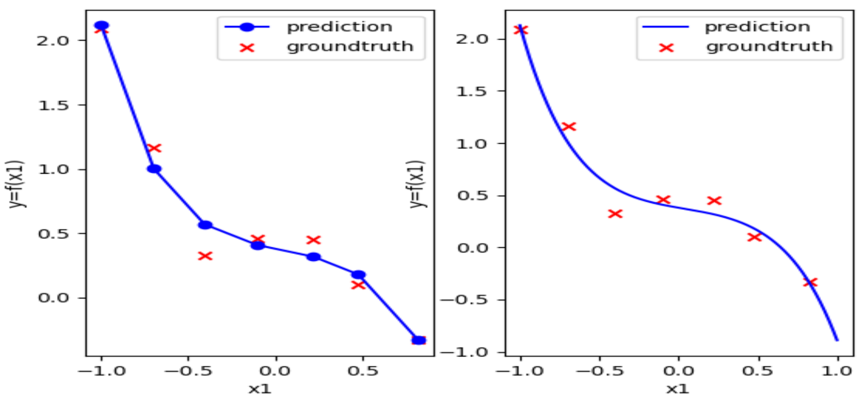
    Minimum cost: 0.00780, on iteration #200



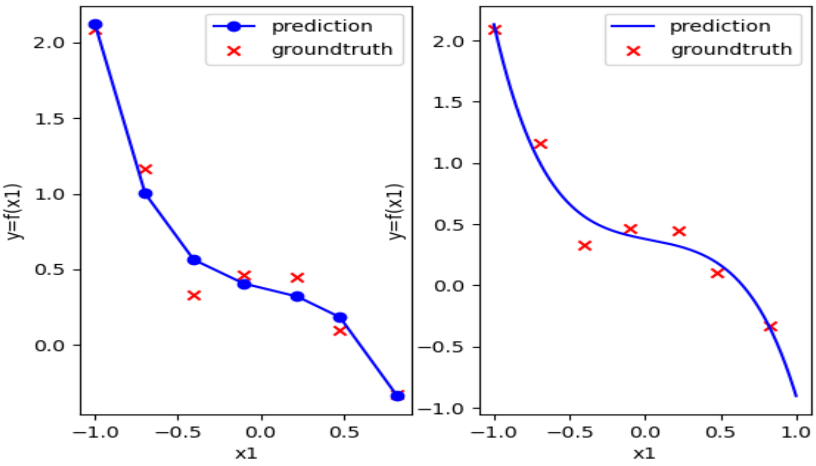


- Experimenting with lambda (l)

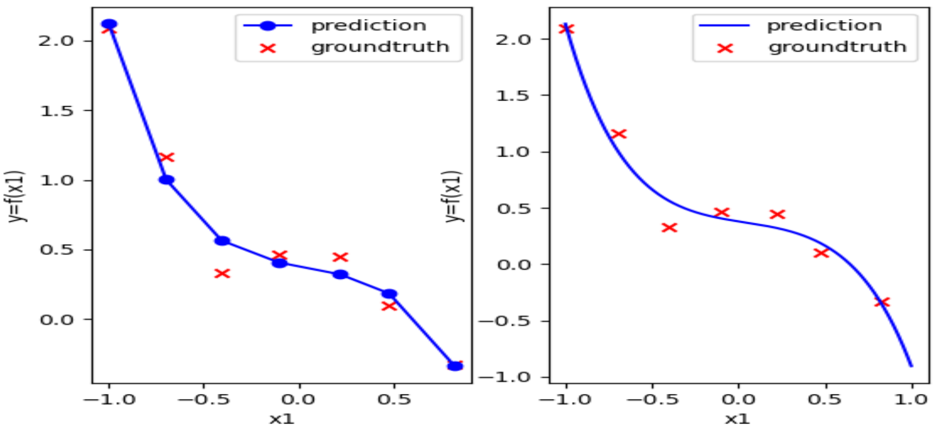
Alpha	Lambda	Minimum Cost
1.0	0.01	0.00865
1.0	0.001	0.00789
1.0	0.0001	0.00781
1.0	0.00001	0.00780
1.0	0.000001	0.00780



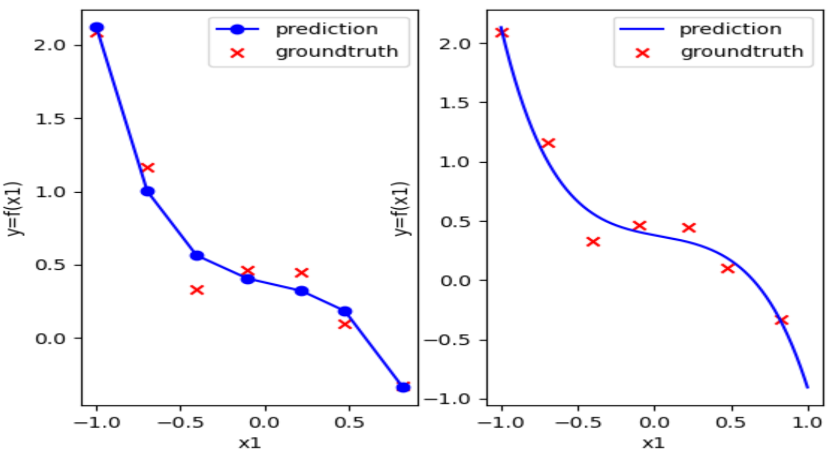
$\lambda = 0.01$, min cost= 0.00865



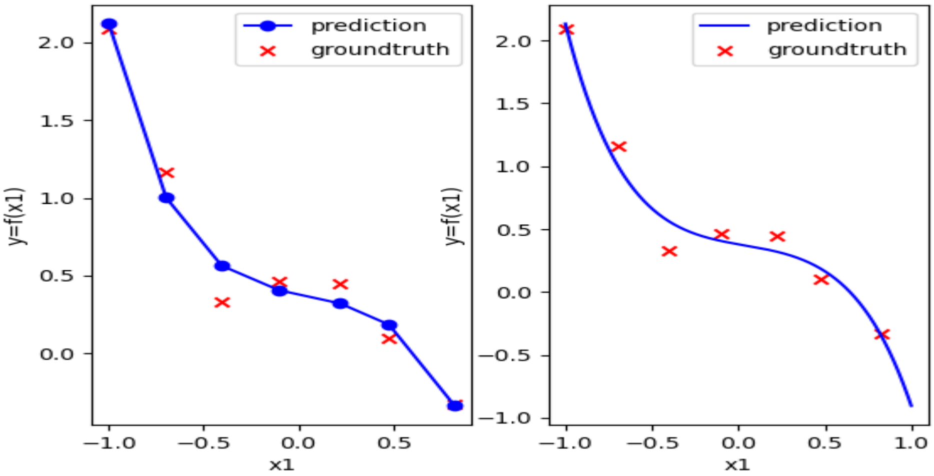
λ = 0.001,   min cost= 0.00789



λ = 0.0001,   min cost= 0.00781



λ =0.00001,   min cost= 0.00780



λ = 0.000001,   min cost = 0.00780

These values are not fixed and trying different combinations with values of alpha, number of iterations and lambda can result in cost going down even further.