Question 1

February 10, 2020

1 COL744 : Machine Learning (Assignment 1)

1.1 Question 1

1.1.1 Part (a) : Implementing Batch Gradient Descent

- In this part I have implemented batch SGD for the loss function $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2$ on the given dataset.
- I have taken η= **0.01**.
- As stopping criteria I am checking **difference of** $J(\theta)$ between two iterations and if it is less than **1e-15** then stopped the iterations. I have also taken an extra parameter **max_iter** = 10^6 describing maximum iterations allowed to bound the number of iterations.
- on given dataset we get $(\boldsymbol{\theta_0}, \boldsymbol{\theta_1})$ = **(0.996620, 0.001340)**

[1]:
```
import numpy as np
import numpy as np

import matplotlib as mp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FuncAnimation
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

from tqdm import tqdm

from sklearn.metrics import mean_squared_error
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler
```

- Loading the data

[2]:
```
X_unnormalized=np.genfromtxt('./ass1_data/data/q1/linearX.csv') #loading data
Y_train=np.genfromtxt('./ass1_data/data/q1/linearY.csv')
```

- Normalizing dataset and adding x_0

```
[3]: X_=(X_unnormalized-X_unnormalized.mean(axis=0))/X_unnormalized.std(axis=0)
X_train=np.vstack((np.ones(X_.shape), X_)).T
```

- Implementing batch gradient descent

```
[4]: def grad(theta, X, Y):
    '''Function to compute partial differentiation wrt theta'''
    err = (X.dot(theta)) - Y #100x1
    loss_val = ((err**2).sum())/(2*X.shape[0])
    grad_val = (1/X.shape[0])*((X.T).dot(err))
    return (grad_val, loss_val)
def LinearRegressionGD(X,Y, r=0.1, max_iter=10**6):
    theta = np.zeros(X.shape[1])
    loss_lst = []
    theta_list=[]
    for i in (range(max_iter)):
        if np.isnan(theta).any():#Checking for divergence
            print('Diverged to infinity')
            return ([],[])
        (grad_val, loss_val) = grad(theta, X, Y)
        theta_next= theta - r * np.array(grad_val)
        theta_list.append(theta)
        loss_lst.append(loss_val)
        if(i>2 and abs(loss_lst[-1]-loss_lst[-2])<1e-15):
            print('converged in %d iterations'%(i))
            break
        theta=theta_next
    return (theta_list, loss_lst)
```

```
[5]: (theta_lst, loss_lst) = LinearRegressionGD(X_train, Y_train, r=0.01,␣
↪max_iter=10**6)
```

```
converged in 1490 iterations
```

```
[6]: theta = theta_lst[-1]
print('theta found by my implementation :(theta0, theta1) = (%f,␣
↪%f)'%(theta[0], theta[1]))
```

```
theta found by my implementation :(theta0, theta1) = (0.996620, 0.001340)
```

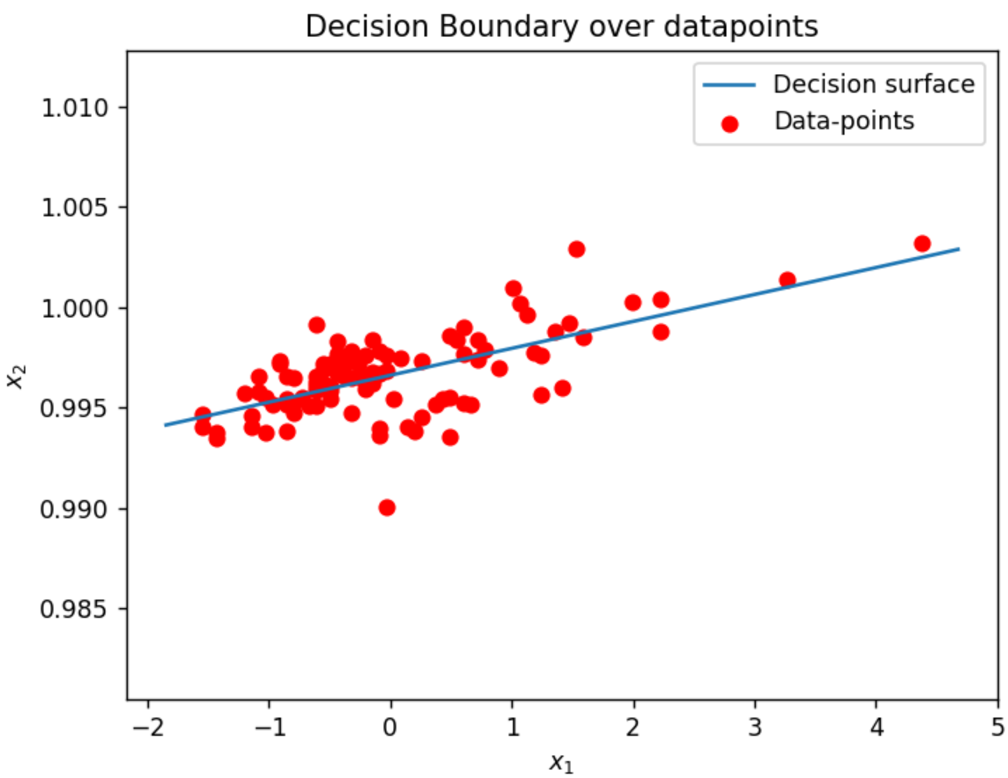
1.1.2 Part (b) Plotting the decision surface

```
[7]: fig, ax = plt.subplots()
ax.scatter(X_train[:,1:], Y_train, c='r', label='Data-points')
axes = plt.gca()
x_vals = np.array(axes.get_xlim())
y_vals = theta[0] + theta[1] * x_vals
```

```python
plt.plot(x_vals, y_vals, label = 'Decision surface')
plt.title('Decision Boundary over datapoints')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.legend()
plt.show()
```

```
&lt;IPython.core.display.Javascript object&gt;
```

```
&lt;IPython.core.display.HTML object&gt;
```



1.1.3 Part (c) : Plotting movement of θ in 3D

3D Plot

[8]:
```python
def plot3d(theta_lst, loss_lst, X_train, Y_train):
    fig = plt.figure(figsize=(8,6))

    # plt.size()
    ax = fig.gca(projection='3d')
```

```
    theta1 = np.arange(-1, 1, 10e-3) #X
    theta0 = np.arange(0, 2, 10e-3) #Y
    X, Y = np.meshgrid(theta1, theta0)
    loss = [[(0.5/X_train.shape[0])*np.linalg.norm((X_train[:,1]*t1 + X_train[:
↪,0]*t0 - Y_train), ord=2)**2 for t1 in theta1] for t0 in theta0]
    Z = np.array(loss).T

    # Plot the surface.
    surf = ax.plot_surface(X, Y, Z, cmap='Reds',
                           linewidth=0, antialiased=False,alpha=0.5)#,␣
↪label='loss surface')
    ax.set_xlabel('$\\theta_1$', color='r')
    ax.set_ylabel('$\\theta_0$', color='r')
    ax.set_zlabel('J($\\theta$)', color='r')

    graph, = plt.plot([], [], 'x',markersize=1, c='black', label = '$<\\theta_1,␣
↪\\theta_0, J(\\theta)>$')
    def animate(i):
        graph.set_data(data[:i+1,1], data[:i+1,0])
        graph.set_3d_properties(loss_data[:i+1])
        return graph

    data = np.array(theta_lst)
    loss_data = np.array(loss_lst)
    anim = FuncAnimation(fig, animate, interval=200)
    plt.legend(loc=4)
    plt.title('3Dplot representing movement of theta ')
    plt.show()
    return anim
```

[11]:
```
anim = plot3d(theta_lst, loss_lst, X_train, Y_train)
```

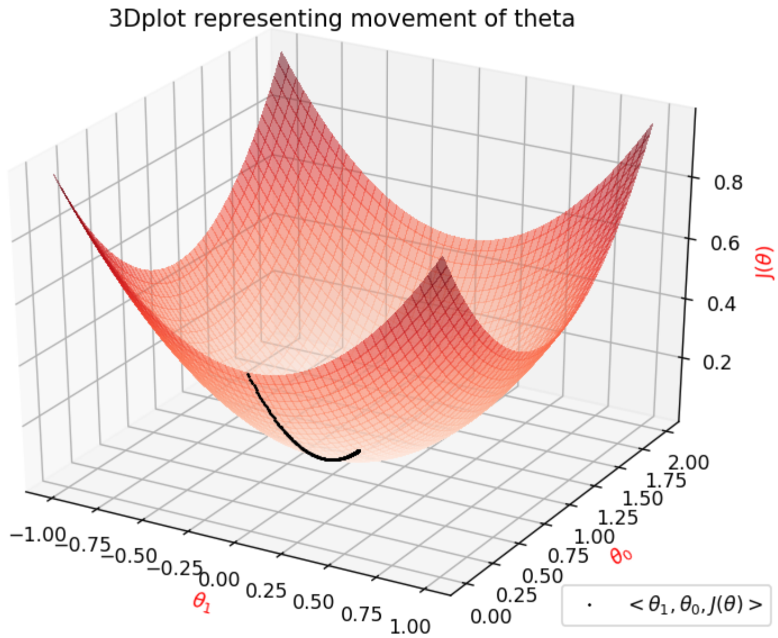
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.1.4 Part (d) : Plotting contour plot representing movement of θ

Contour Plot

[12]:
```
def plot_contour(theta_lst, X_train, Y_train, learning_rate):
    plt.rcParams.update({'font.size': 7})
    theta1 = np.arange(-1, 1, 10e-2) #X
    theta0 = np.arange(-0.5, 1.5, 10e-2) #Y
    X, Y = np.meshgrid(theta1, theta0)
```




```
    loss = [[np.linalg.norm((X_train[:,1]*t1 + np.ones(X_train.shape[0])*t0 -␣
↪Y_train), ord=2)**2 for t1 in theta1] for t0 in theta0]
    Z = np.array(loss)

    fig, ax = plt.subplots()
    CS = ax.contour(X, Y, Z)
    ax.clabel(CS, inline=1, fontsize=5)
    ax.set_title('countour plot representing movement of theta in each␣
↪iterations for $\\eta$ = %s'%(learning_rate))
    ax.set_xlabel('$\\theta_1$', color='r')
    ax.set_ylabel('$\\theta_0$', color='r')

    graph, = plt.plot([], [], 'x',markersize=1, label = '$<\\theta_1,␣
↪\\theta_0>$')

    def animate(i):
        graph.set_data(data[:i+1, 1:2], data[:i+1, 0:1])
        return graph

    data = np.array(theta_lst)
    anim = FuncAnimation(fig, animate, interval=100)
    plt.legend()
```

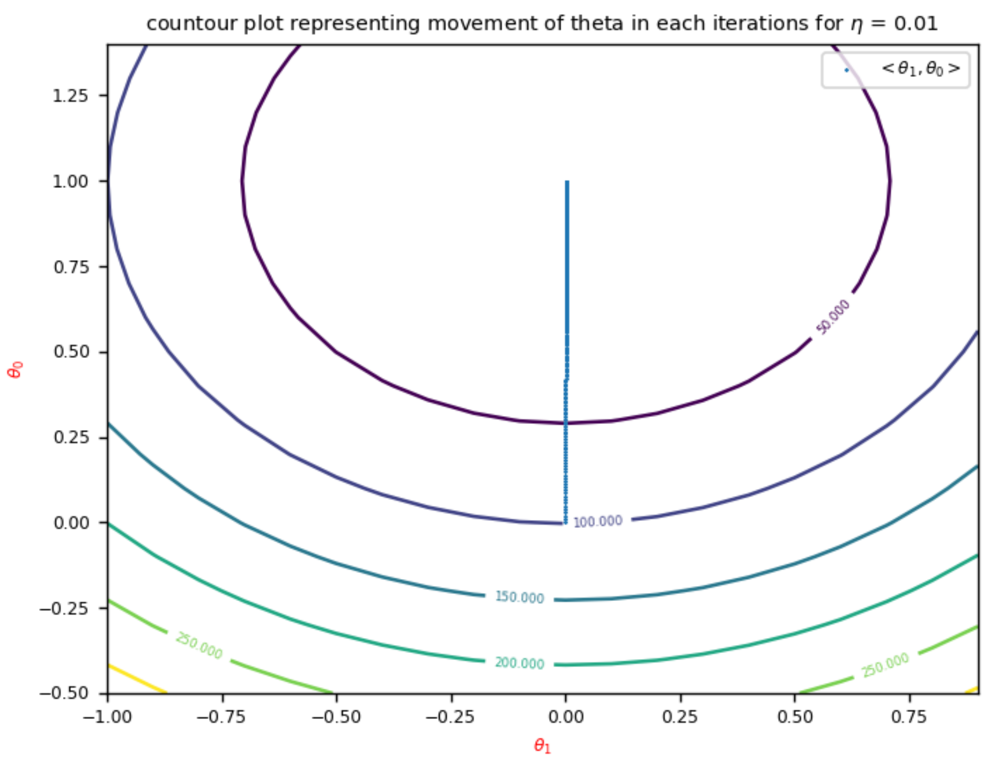
```
    plt.show()
    return anim
```

[11]:
```
anim = plot_contour(theta_lst, X_train, Y_train, 0.01)
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[11]: <matplotlib.animation.FuncAnimation at 0x1dde8018988>



1.1.5 Part (e) : Comparing various learning rate and plotting Contour Plot for them

- For η=**0.001**

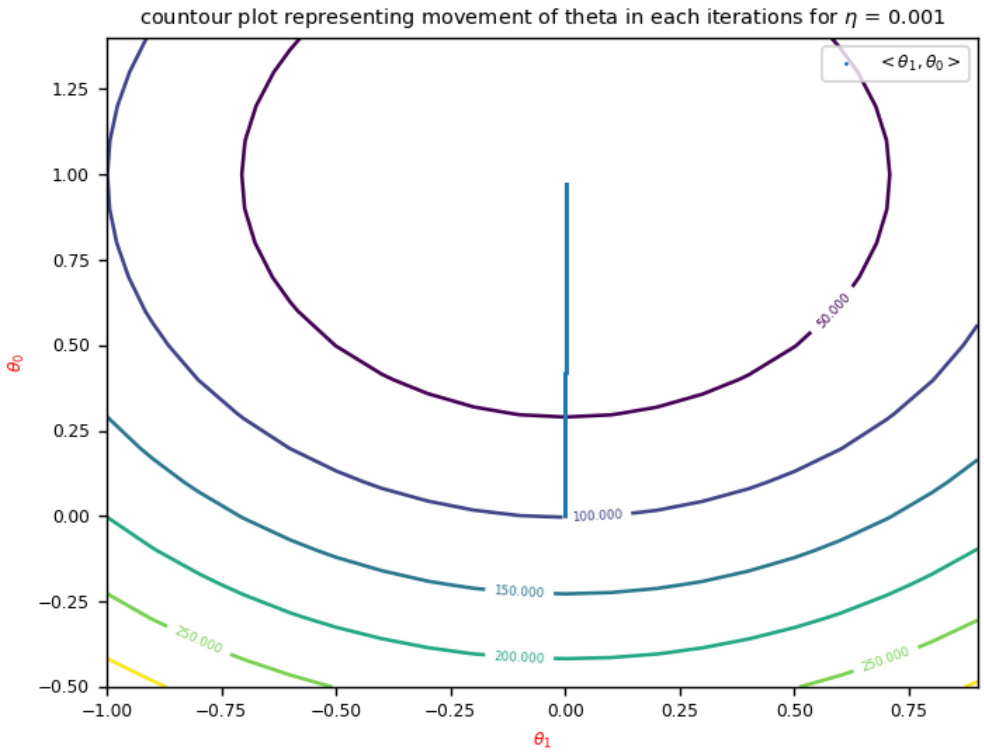
[12]:
```
(theta_lst, loss_lst) = LinearRegressionGD(X_train, Y_train, r=0.001,␣
 ↪max_iter=10**6)
plot_contour(theta_lst, X_train, Y_train, 0.001)
```

converged in 13806 iterations

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[12]: <matplotlib.animation.FuncAnimation at 0x1dde80dde88>



- For η=**0.025**

[13]:
```
(theta_lst, loss_lst) = LinearRegressionGD(X_train, Y_train, r=0.025,
    max_iter=10**6)
plot_contour(theta_lst, X_train, Y_train, 0.025)
```

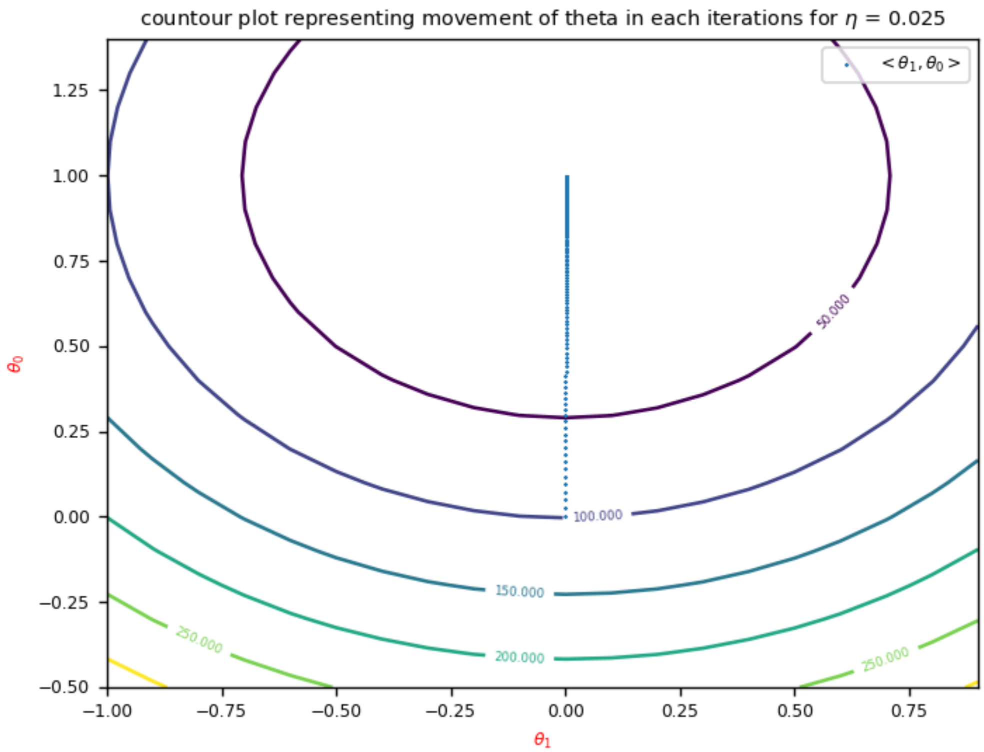
converged in 610 iterations

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

[13]: <matplotlib.animation.FuncAnimation at 0x1dde8a298c8>

- For η=**0.1**




```
[14]: (theta_lst, loss_lst) = LinearRegressionGD(X_train, Y_train, r=0.1,␣
       ↪max_iter=10**6)
      plot_contour(theta_lst, X_train, Y_train, 0.1)
```

```
converged in 154 iterations

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

```
[14]: <matplotlib.animation.FuncAnimation at 0x1dde7d4b388>
```

1.1.6 Observations

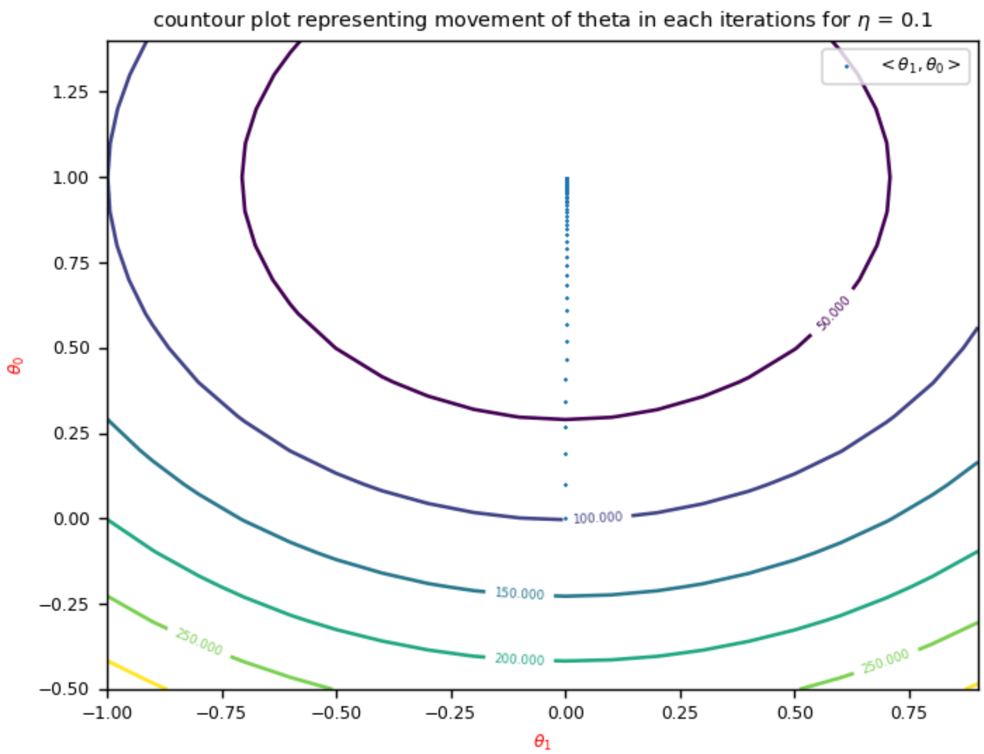
- Here I have mentioned number of iterations per eta in the table below

η	Number of Iterations
0.01	1490
0.001	13806
0.025	610
0.1	154

η	Number of Iterations

Here in this table we can see that number of iterations decreases as be increase eta as the difference would be higher.

- Also in contour plot we can notice that for large η like 0.1 we see big jumps in starting and as θ converges to θ^* we can see smaller jumps, whereas in smaller θ like 0.001 we can see that from the beginning it is taking very small jumps and relatively it also takes more time for θ to reach to optimal value as number of iterations was very large in comparision.


countour plot representing movement of theta in each iterations for η = 0.1
$<\theta_1, \theta_0>$
50.000
100.000
150.000
200.000
250.000
250.000
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
θ_0
θ_1