Machine Learning
Assignment -01

February 10, 2020

Linear Regression

[1]:
```
# %matplotlib notebook
import seaborn as sb
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import threading
import time
from matplotlib import cm
import os
notebook_path = os.path.abspath("q1.ipynb")
```

[2]:
```
def normalization(X):
    X_mean = np.mean(X)
    X_var= np.sum((X-X_mean)**2)/len(X)
    X_std_dev= np.sqrt(X_var)
    X_norm = (X- X_mean)/X_std_dev
    return X_norm

def cost(X1, theta1, Y1):
    return np.sum((np.dot(X1, theta1) - Y1) ** 2) /(2* len(Y1))

def grad_cost(X,theta, Y):
    temp = Y-np.dot(X,theta)
    temp2= temp * -X
    theta = (temp2.sum(axis=0))/(len(Y))
    return theta.T.reshape((theta.T.shape[0], 1))

def linear_reg(X_norm,Y, eta, thetas):
    theta = np.zeros((2, 1))
    thetas.append(theta[0][0])
    thetas.append(theta[1][0])
```

```
    count=0
    prev_cost=cost(X_norm, theta, Y)
    for i in range(100000):
        count+=1
        gcost=grad_cost(X_norm, theta, Y)
        theta = theta - eta * gcost
        curr_cost =  cost(X_norm, theta, Y)
        if abs(prev_cost-curr_cost)<= 10^-4:
            break
        prev_cost= curr_cost
        if i%100==0 :
            thetas.append(theta[0][0])
            thetas.append(theta[1][0])
    return theta,count

def plot_hypothesis(X_norm,Y, res):
    sb.set()
    fig, ax = plt.subplots(figsize=(8, 6), dpi= 80)
    a=X_norm[:,1:2]
    ax.plot(a,Y,'b^')
    plt.xlabel('Acidity(x1)')
    plt.ylabel('Density(h(x))')
    plt.title('x-y')
    Y_dash =np.dot(X_norm,res)
    plt.plot(a,Y_dash,color='red')
    plt.show();
    return fig,ax
```

[3]:
```
def animate(i, th0, th1, actual_cost,line):
    line.set_data(th0[:i],th1[:i])
    line.set_3d_properties(actual_cost[:i])
    return line,

def plot_surface(X_norm,Y, th0, th1, actual_cost):
    theta0 = np.linspace(0,2, 100)
    theta1 = np.linspace(-1,1,100)
    Theta0, Theta1 = np.meshgrid(theta0, theta1)
    zs = np.array([cost(X_norm,np.array([[i],[j]]),Y) for i,j in zip(np.
 ↪ravel(Theta0), np.ravel(Theta1))])
    Cost = zs.reshape(Theta0.shape)

    #plotting
    fig = plt.figure(figsize=(8, 6), dpi= 80)
    ax = plt.axes(projection='3d')
    ax.plot_surface(Theta0, Theta1, Cost, cmap=cm.
 ↪coolwarm,linewidth=0,antialiased=False, alpha=0.8)
    line, = ax.plot([],[],[],lw=2)
```

```
    ax.set_xlabel('theta0')
    ax.set_ylabel('theta1')
    ax.set_zlabel('J(theta)')
#     anim = animation.FuncAnimation(fig, animate,frames=len(th0), fargs=(th0,␣
↪th1, actual_cost, line), interval=200,
#                               repeat_delay=1, blit=True)
    line.set_data(th0,th1)
    line.set_3d_properties(actual_cost)
    plt.show();
    return plt
```

[4]:
```
def animate2(i, th0, th1, line1):
    line1.set_data(th0[:i],th1[:i])
    return line1,

def plot_contour(X_norm,Y,th0, th1, actual_cost, fig_no):
    fig2 = plt.figure(figsize=(8, 6), dpi= 80)
    ax=plt.axes()
    theta0 = np.linspace(0,2, 100)
    theta1 = np.linspace(-1,1,100)
    Theta0, Theta1 = np.meshgrid(theta0, theta1)
    zs = np.array([cost(X_norm,np.array([[i],[j]]),Y) for i,j in zip(np.
↪ravel(Theta0), np.ravel(Theta1))])
    Cost = zs.reshape(Theta0.shape)
    ax.contour(Theta0, Theta1, Cost)
    ax.set_xlabel('theta0')
    ax.set_ylabel('theta1')
    line1, = ax.plot([],[],lw=1.5)
    line1.set_data(th0,th1)
#     anim2 = animation.FuncAnimation(fig2, animate2,frames=200, fargs=(th0,␣
↪th1,line1),interval=200,repeat_delay=5, blit=True)
    plt.show();
    return ax
```

[5]:
```
with open('data/q1/linearX.csv') as fp:
    x=[]
    for line in fp:
        x.append(float(line[:-1]))
with open('data/q1/linearY.csv') as fp:
    y=[]
    for line in fp:
        y.append(float(line[:-1]))

X1=np.array(x)
Y1=np.array(y)
X = X1.reshape((X1.shape[0], 1))
Y = Y1.reshape((Y1.shape[0], 1))
```

```
X_norm1= normalization(X)
X_norm = np.append(np.ones((X_norm1.shape[0], 1)), X_norm1, axis = 1)
theta = np.zeros((2, 1))
```

1(a)

Parameters:

$\theta_0 = 0.9966201$

$\theta_1 = 0.0013402$

Learning Rate - 0.02

Stopping Criteria when difference in cost is less than 10^{-4}

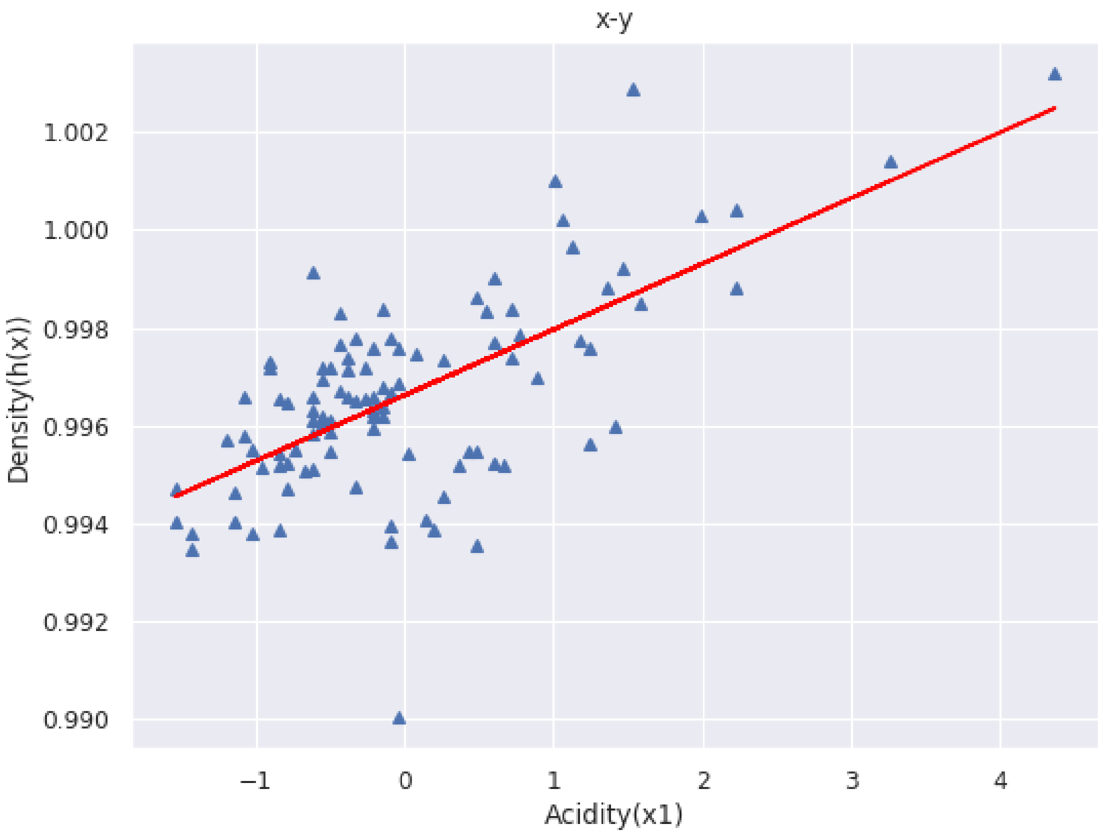
$J(\theta^{(i)}) - J(\theta^{(i+1)}) <= 0.0001$

[6]:
```
thetas= []
eta =0.02
res, itr = linear_reg(X_norm, Y,eta, thetas)
print("parameters")
print(res)
print("cost", cost(X_norm,res,Y))
thetas1=np.array(thetas)
thetas = thetas1.reshape(int(thetas1.shape[0]/2), 2)

th0=[thetas[i][0] for i in range(thetas.shape[0])]
th1=[thetas[i][1] for i in range(thetas.shape[0])]
actual_cost= np.array([cost(X_norm, np.reshape(thetas[i],(2,1)), Y) for i in␣
↪range(thetas.shape[0])])
```

```
parameters
[[0.9966201]
 [0.0013402]]
cost 1.1947898109836582e-06
```

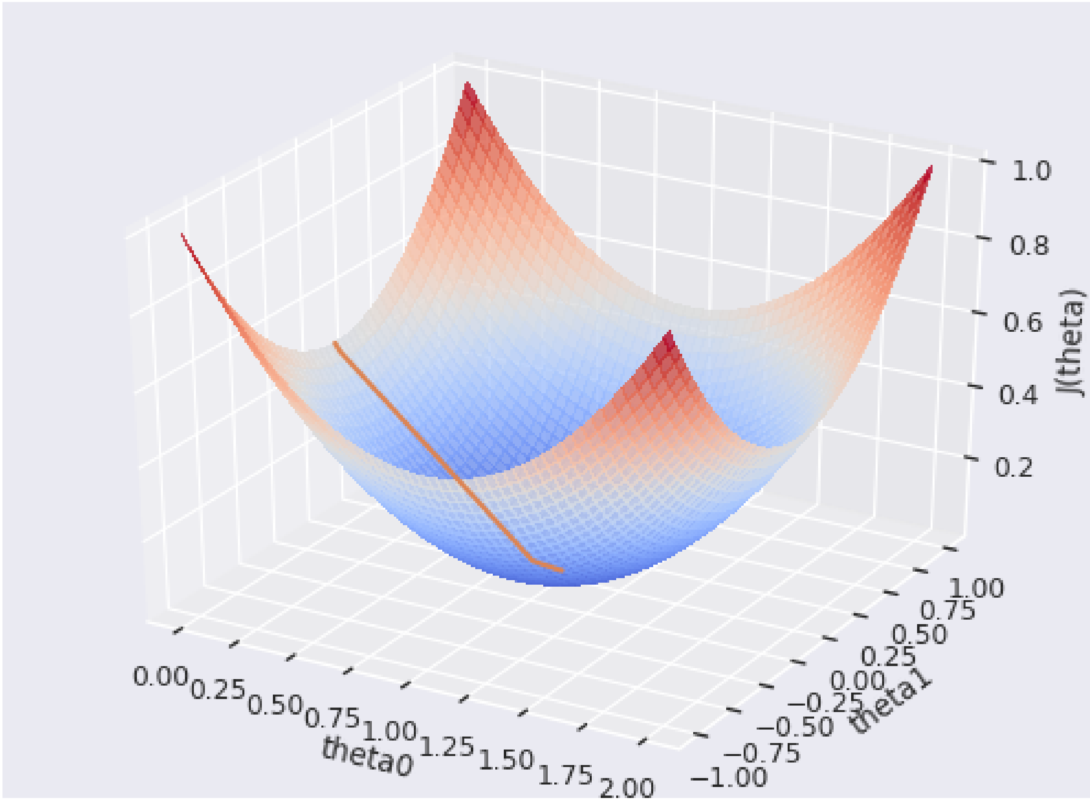
(b) Plot of Hypothesis Function

[7]:
```
plot_hypothesis(X_norm, Y, res)
```

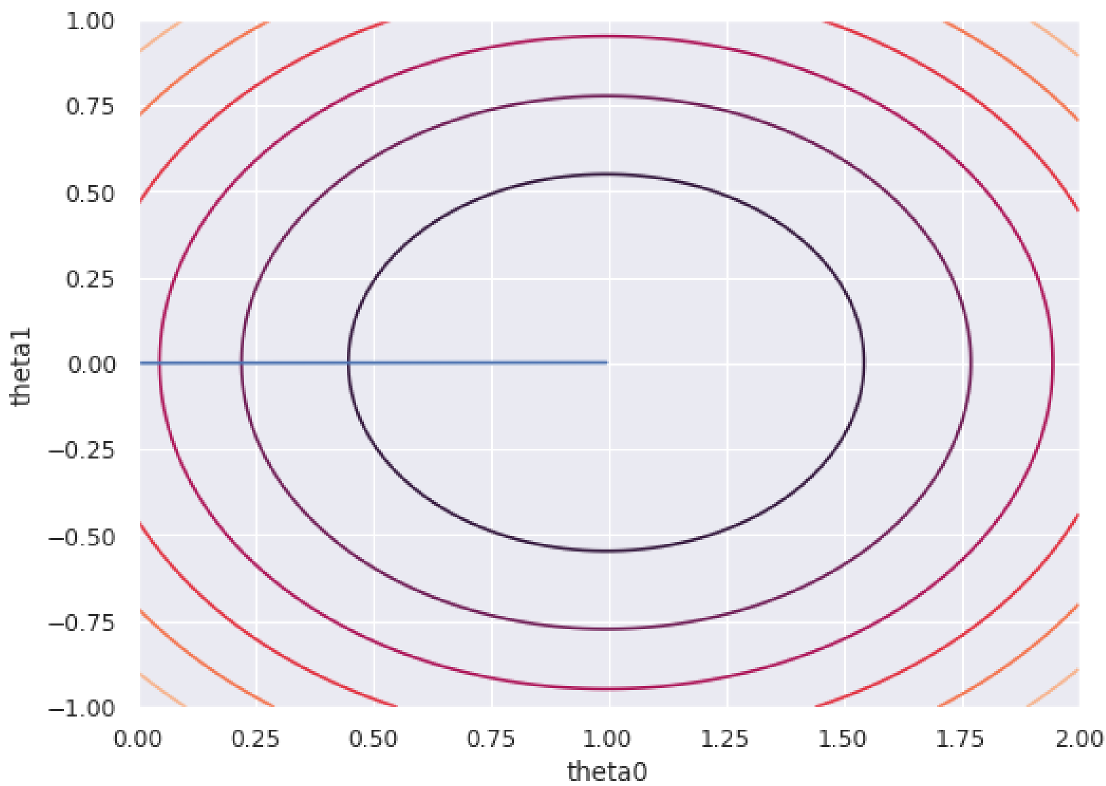


(c) Cost Function

[8]:
```
plot_surface(X_norm, Y,th0, th1, actual_cost)
```



[9]:
```
plot_contour(X_norm,Y,th0, th1, actual_cost,2)
```



(d) Contours of error function

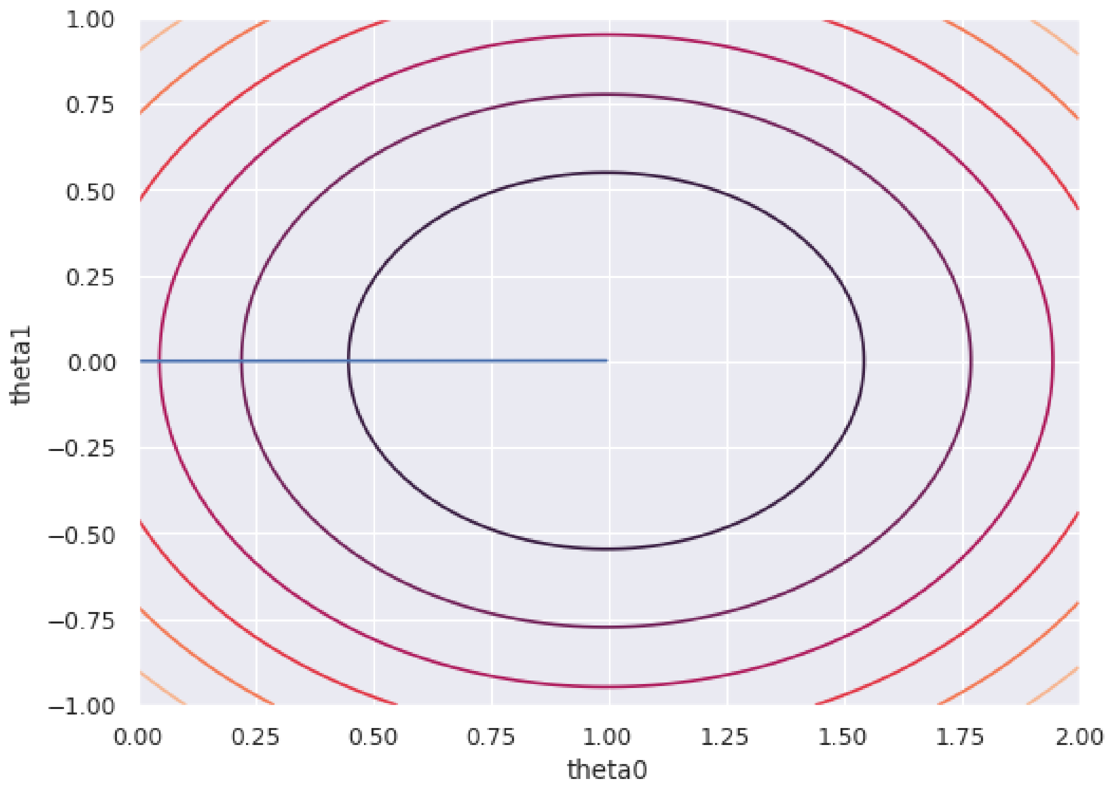
(e) Learning Rate = 0.001

[10]:
```
thetas= []
eta =0.001
#linear Regression
start = time.time()
res, itr = linear_reg(X_norm, Y,eta, thetas)
print("parameters")
print(res)
print("cost", cost(X_norm,res,Y))
print('time', time.time()-start)
thetas1=np.array(thetas)
thetas = thetas1.reshape(int(thetas1.shape[0]/2), 2)

th0=[thetas[i][0] for i in range(thetas.shape[0])]
th1=[thetas[i][1] for i in range(thetas.shape[0])]
actual_cost= np.array([cost(X_norm, np.reshape(thetas[i],(2,1)), Y) for i in␣
↪range(thetas.shape[0])])
```

```

```

```
parameters
[[0.9966201]
 [0.0013402]]
cost 1.1947898109836603e-06
time 7.714576959609985
```

[11]:
```
plot_contour(X_norm,Y,th0, th1, actual_cost,2)
```



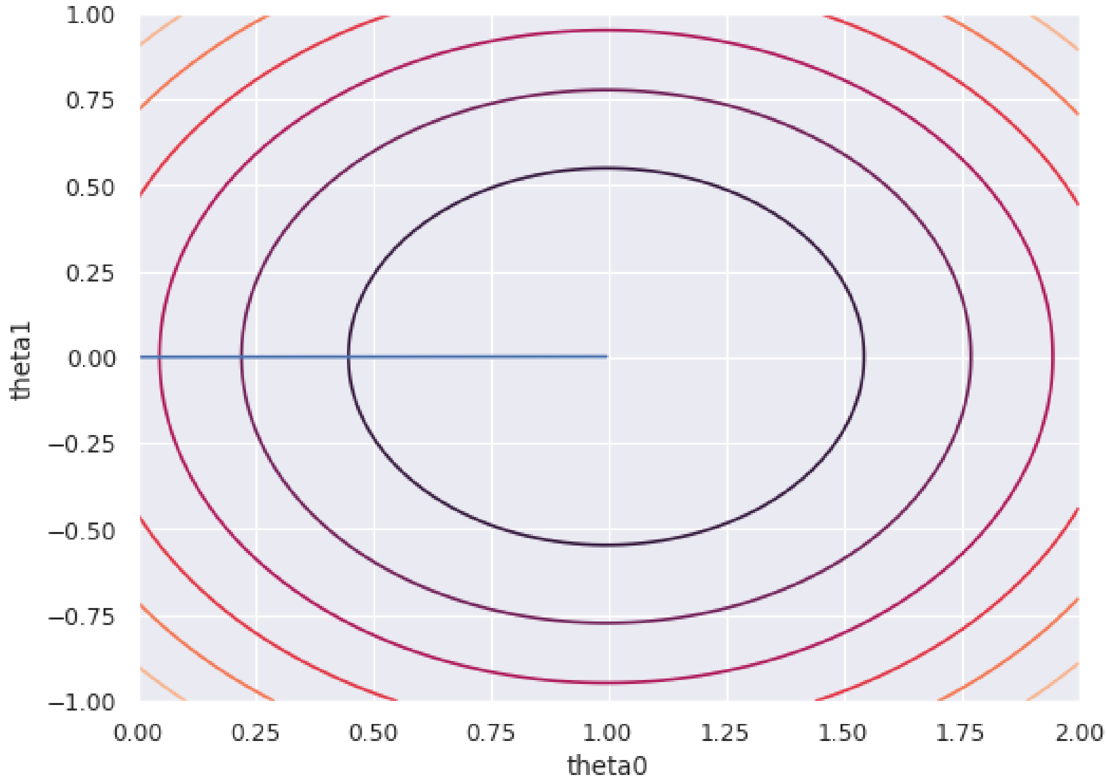
Learning Rate = 0.025

[12]:
```
thetas= []
eta =0.025
start = time.time()
res, itr = linear_reg(X_norm, Y,eta, thetas)
print("parameters",res.reshape(1,2))
print("cost", cost(X_norm,res,Y))
print('time-', time.time()-start )
thetas1=np.array(thetas)
```

```
thetas = thetas1.reshape(int(thetas1.shape[0]/2), 2)
th0=[thetas[i][0] for i in range(thetas.shape[0])]
th1=[thetas[i][1] for i in range(thetas.shape[0])]
actual_cost= np.array([cost(X_norm, np.reshape(thetas[i],(2,1)), Y) for i in␣
 ↪range(thetas.shape[0])])
```

```
parameters [[0.9966201 0.0013402]]
cost 1.194789810983658e-06
time- 6.560656309127808
```

[13]:
```
plot_contour(X_norm,Y,th0, th1, actual_cost,3)
```



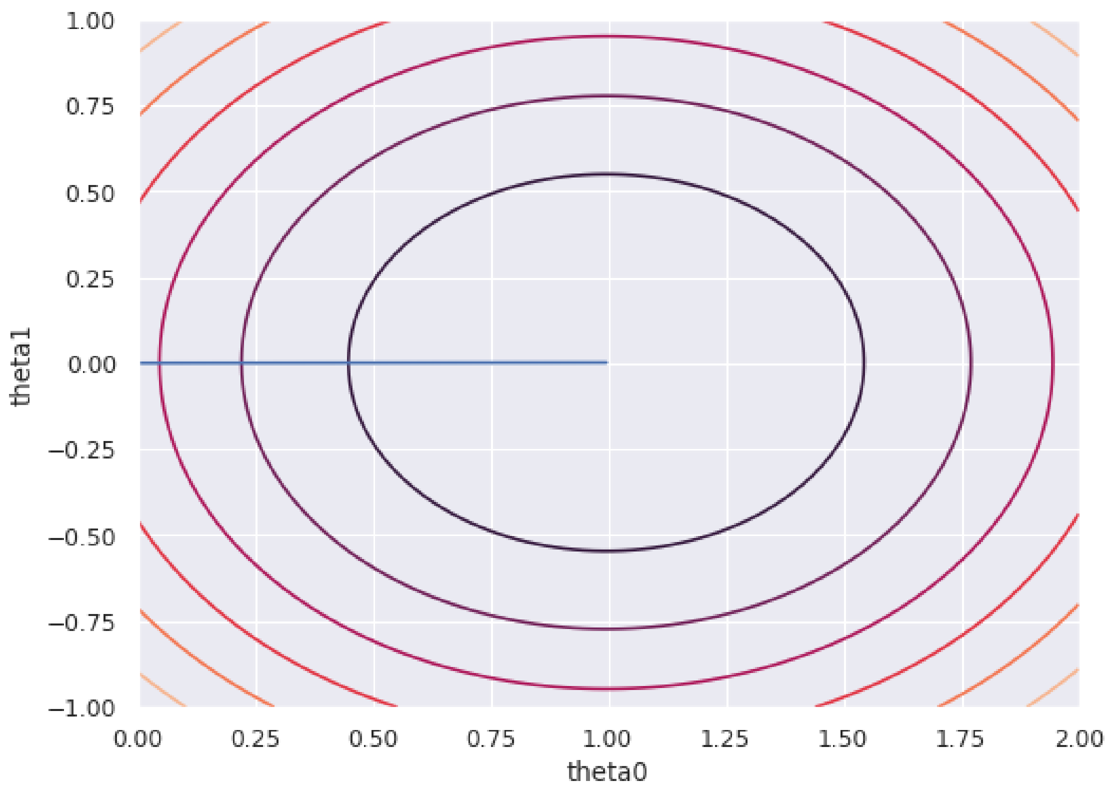
Learning Rate = 0.1

[14]:
```
thetas= []
eta =0.1
start = time.time()
res, itr = linear_reg(X_norm, Y,eta, thetas)
print("parameters")
print(res)
```

```
print("cost", cost(X_norm,res,Y))
print('time', time.time()-start)
thetas1=np.array(thetas)
thetas = thetas1.reshape(int(thetas1.shape[0]/2), 2)

th0=[thetas[i][0] for i in range(thetas.shape[0])]
th1=[thetas[i][1] for i in range(thetas.shape[0])]
actual_cost= np.array([cost(X_norm, np.reshape(thetas[i],(2,1)), Y) for i in␣
 ↪range(thetas.shape[0])])
```

```
parameters
[[0.9966201]
 [0.0013402]]
cost 1.1947898109836607e-06
time 6.053455591201782
```

[15]:
```
plot_contour(X_norm,Y,th0, th1, actual_cost,4)
```



(e) Observation

When learning rate is low it takes more time to converge as compared to high learning rate but learning rate can be at certain limits as at 2.5 or after that it starts diverging . Time taken in each case

Learning rate time taken

0.001 7.14042186737060\55

0.025 7.627198219299316

0.1 6.640247821807861

Question2

0.1 Question(2) Stochastic Gradent Descent

[1]:
```
import seaborn as sb
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import threading
import time
from matplotlib import cm
```

0.2 (a) Sampling

Successfully sampled data with given normal distribution.

$x1 \sim N(3,4)$

$x2 \sim N(-1,4)$

[42]:
```
with open('data/q2/q2test.csv') as fp:
    x_test=[]
    y_test=[]
    for line in fp:
        line=line[:-1]
        [x_t1,x_t2,y_t]=line.split(',')
        x_test.append(float(x_t1))
        x_test.append(float(x_t2))
        y_test.append(float(y_t))
X_test1 = np.array(x_test).reshape(int(len(x_test)/2),2)
X_test = np.append(np.ones((X_test1.shape[0], 1)), X_test1, axis=1)
Y_test= np.array(y_test).reshape(Y_test.shape[0],1)

x1= np.random.normal(3, 2, 1000000)
x2= np.random.normal(-1, 2, 1000000)
e= np.random.normal(0, np.sqrt(2), 1000000)
Y1= 3+ x1+ 2*x2+ e
```

```
Y = Y1.reshape(Y1.shape[0],1)

X_temp = np.append(np.ones((1000000,1)), x1.reshape(1000000,1) ,axis=1)
X_temp2 =np.append(X_temp, x2.reshape(1000000,1), axis=1)

X_Y= np.append(X_temp2, Y, axis=1)
np.random.shuffle(X_Y)
X= X_Y[:,0:3]
Y=X_Y[:,3:4]
```

0.3 (b) Implementation of Stochastic Gradent Descent

[43]:
```
def cost(X1, theta1, Y1):
    return (np.sum((np.dot(X1, theta1) - Y1) ** 2) /(2* len(Y1)))



def grad_cost(X,theta, Y):
    temp = Y-np.dot(X,theta)
    temp2= temp * -X
    theta = (temp2.sum(axis=0))/(len(Y))
    return theta.T.reshape((theta.T.shape[0], 1))



def sto_gradient(X,Y,thetas,r, diff, steps, avg_no):
    total_itr=flag =0
    avg_th=[]
    theta = np.zeros((3, 1))
    prev_cost = cost(X[:r,:], theta, Y[:r,:])
    for  j in range(steps):
        for i in range(0,1000000,r):
            total_itr =total_itr+1
            X_batch = X[i:r+i,:]
            Y_batch= Y[i:r+i,:]
            gcost=grad_cost(X_batch, theta, Y_batch)
            theta = theta - 0.001 * gcost
            if i%(r*10)==0 :
                thetas.append(theta[0][0])
                thetas.append(theta[1][0])
                thetas.append(theta[2][0])
            curr_cost=  cost(X_batch, theta, Y_batch)
            avg_th.append(curr_cost)
            if(total_itr % avg_no == 0):
                cost_diff = prev_cost-np.mean(avg_th)
                prev_cost = np.mean(avg_th)
                avg_th =[]
```

```
                if abs(cost_diff) <= diff and total_itr > 1000000/r:
                    flag=1
                    break
        if flag==1:
            break
    return theta ,total_itr, thetas
```

[44]:
```
def animate(i, th0, th1, th2,line):
    line.set_data(th0[:i],th1[:i])
    line.set_3d_properties(th2[:i])
    return line,

def plot(thetas1):
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    thetas=np.array(thetas1)
    thetas = thetas.reshape(int(thetas.shape[0]/3), 3)
    th0=[thetas[i][0] for i in range(thetas.shape[0])]
    th1=[thetas[i][1] for i in range(thetas.shape[0])]
    th2=[thetas[i][2] for i in range(thetas.shape[0])]
    ax.set_xlim3d(0, 3.2)
    ax.set_ylim3d(0,1.5)
    ax.set_zlim3d(0,2)
    line, = ax.plot([],[],[],lw=2)
    line.set_data(th0,th1)
    line.set_3d_properties(th2)
    ax.set_xlabel('theta0')
    ax.set_ylabel('theta1')
    ax.set_zlabel('theta3')
#     anim = animation.FuncAnimation(fig, animate,frames=len(th0), fargs=(th0,␣
 ↪th1, th2, line), interval=40,
#                                  repeat_delay=1, blit=True)
    plt.show();
    return ax
```

0.4 Batch size r =1

parameters learned

$\theta_0 = 2.9480744$

$\theta_1 = 1.01232689$

$\theta_2 = 2.01740026$

Diffrence in parameters $\theta\_0 = 0.0519256$

$\theta_1 = 0.01232688$

$\theta_2 = 0.0174$

Time Taken 69.81 seconds

N0. of iterations- 1064000

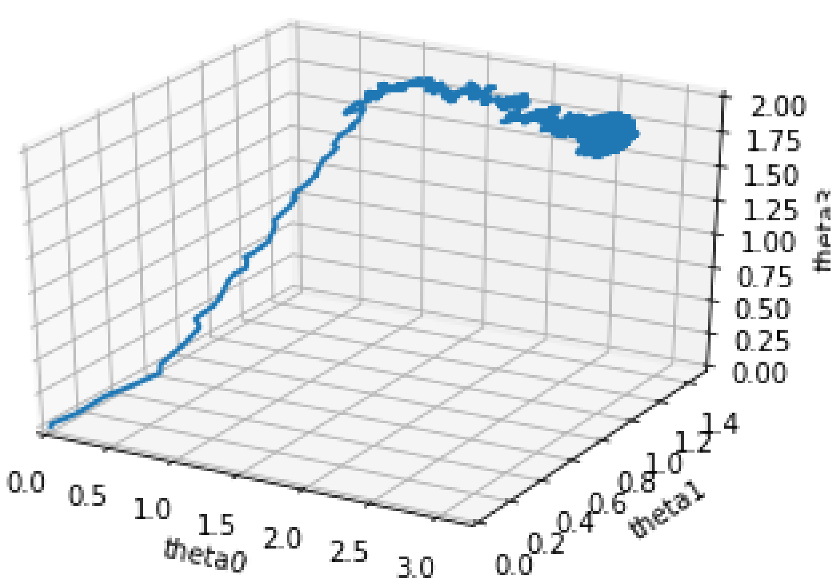
Cost - 1.002598692

Convergence Criteria First traverse the whole m examples atleast once then taken average cost after every 1000 iterations an d compare it with prev average cost
if $J(\theta^{(i)}) - J(\theta^{(i+1)}) <= 0.0001$ then converged Also set upper bound on number of iterations of data so if it doesn't pass convergence criteria then it will stop after 1000

[5]:
```
theta= np.ones((3,1))
thetas= [0,0,0]
start_time =time.time()
res1, itr, thetas = sto_gradient(X,Y,thetas,1,0.0001,1000,1000)
print('Time',time.time()-start_time)
print('Iterations- ',itr)
print(res1)
print('cost ',cost(X,res1, Y))
print('Done')
```

```
Time 69.81869792938232
Iterations-  1064000
[[2.9480744 ]
 [1.01232689]
 [2.01740026]]
cost  1.0025986920777687
Done
```

[6]:
```
plot(thetas)
```



[6]: <matplotlib.axes._subplots.Axes3DSubplot at 0x7f0940cbb3d0>

0.5 Batch size r =100

parameters learned

$\theta\_0 = 2.99737108$

$\theta_1 = 0.99731907$

$\theta_2 = 2.00097804$

Difference in parameters

$\theta_0 = 0.00262891$

$\theta_1 = 0.00268$

$\theta\_2 = 0.00097804$

Time Taken 86.4108 seconds

N0. of iterations- 1000000

Cost - 1.0013732546240643

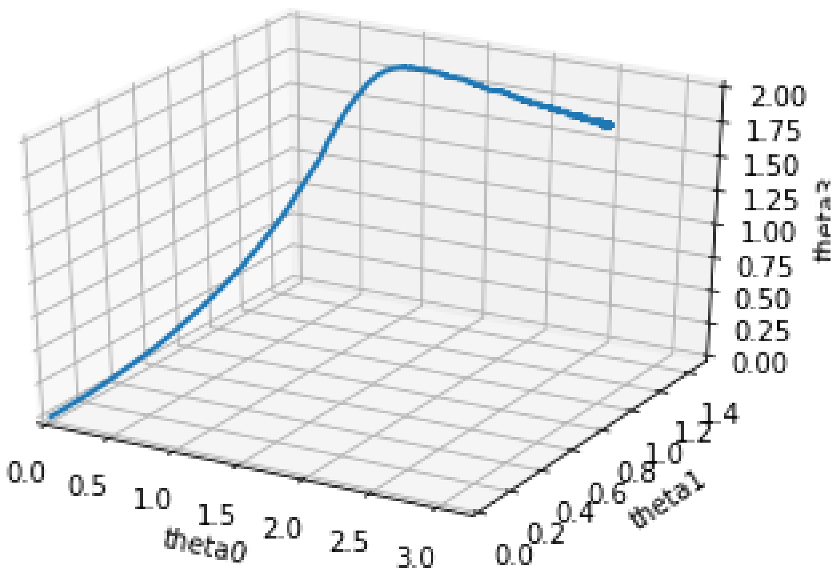
Convergence criteia
First traverse the whole m examples atleast once then taken average cost after every 1000 iterations an d compare it with prev average cost
if $J(\theta^{(i)}) - J(\theta^{(i+1)}) <= 0.0001$ then converged

[7]:
```
theta= np.ones((3,1))
thetas= [0,0,0]
start_time =time.time()
# sto_gradient(X,Y,thetas,r, diff, steps, avg_no)
res2, itr, thetas = sto_gradient(X,Y,thetas,100,0.0001,100, 1000)
print('start')
print('Time',time.time()-start_time)
print('Iterations- ',itr)
print(res2)
print('cost ',cost(X,res2, Y))
print('Done')
```

```
start
Time 86.41087055206299
Iterations-  1000000
[[2.99737108]
 [0.99731907]
 [2.00097804]]
```

```
cost  1.0013732546240643
Done
```

[8]:
```
plot(thetas)
```



[8]:
```
<matplotlib.axes._subplots.Axes3DSubplot at 0x7f0942047550>
```

0.6 Batch size r =10000

parameters learned

$\theta_0 = 2.96347761$

$\theta_1 = 1.01232689$

$\theta_2 = 0.01740026$

Difference in parameters

$\theta_0 = 0.03652239$

$\theta_1 = 0.01232688$

$\theta\_2 = 0.00097804$

Time Taken 19.9656553 seconds

N0. of iterations- 1064000

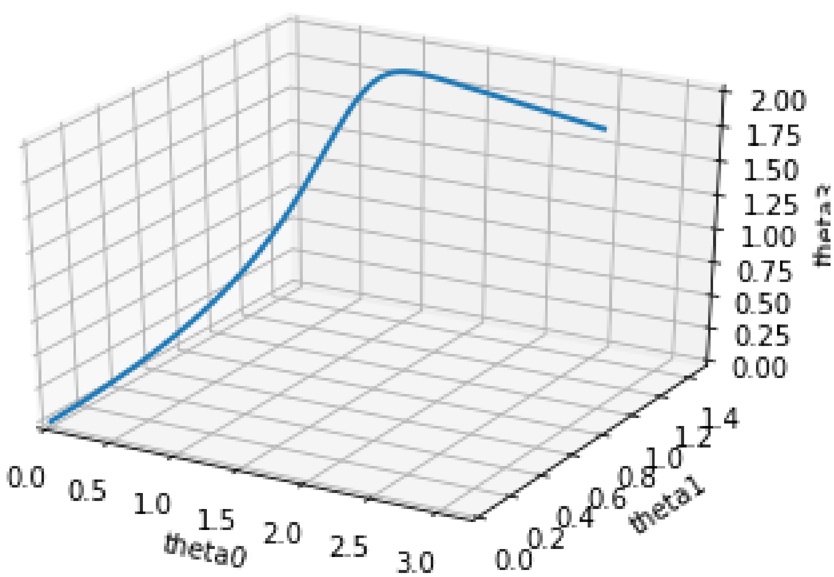
Error - 1.002598692

Convergence Criteria First traverse the whole m examples atleast once then taken average cost after every 1000 iterations an d compare it with prev average cost
if $J(\theta^{(i)}) - J(\theta^{(i+1)}) <= 0.00001$ then converged

[9]:
```
theta= np.ones((3,1))
thetas= [0,0,0]
start_time =time.time()
# sto_gradient(X,Y,thetas,r, diff, steps, avg_no)
res3, itr, thetas = sto_gradient(X,Y,thetas,10000,0.00001,1000, 100)
print('start')
print('Time',time.time()-start_time)
print('Iterations- ',itr)
print(res3)
print('cost ',cost(X,res3, Y))
print('Done')
```

```
start
Time 19.96565532684326
Iterations-  16200
[[2.96347761]
 [1.00775299]
 [1.99844067]]
cost  1.0014768467189297
Done
```

[10]:
```
plot(thetas)
```



[10]: <matplotlib.axes._subplots.Axes3DSubplot at 0x7f09419d3850>

Parameters Learned $\theta\_0 =$

0.7 Batch size r =1000000

parameters learned

$\theta_0 = 2.890994271$
$\theta_1 = 1.02368121$
$\theta_2 = 1.99112236$

Time Taken 19.9656553 seconds

N0. of iterations- 11770

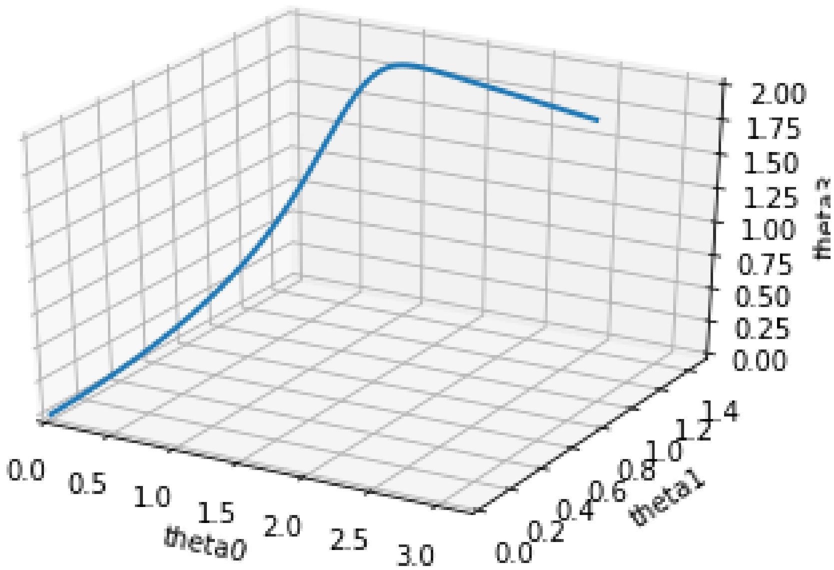
Error - 1.00052844954

Convergence Criteria First traverse the whole m examples atleast once then taken average cost after every 1000 iterations an d compare it with prev average cost
if $J(\theta^{(i)}) - J(\theta^{(i+1)}) <= 10^{-6}$ then converged

[34]:
```
theta= np.ones((3,1))
thetas= [0,0,0]
start_time =time.time()
# sto_gradient(X,Y,thetas,r, diff, steps, avg_no)
print('start')
res4, itr , thetas= sto_gradient(X,Y,thetas,1000000,0.000001,50000, 1)
print('Time',time.time()-start_time)
print('Iterations- ',itr)
print(res4)
print('cost ',cost(X,res4, Y))
print('Done')
```

```
start
Time 1331.3197798728943
Iterations-  11770
[[2.89099427]
 [1.02368121]
 [1.99112236]]
cost  1.0005284495402436
Done
```

[45]:
```
plot(thetas)
```



[45]: <matplotlib.axes._subplots.Axes3DSubplot at 0x7f093e299c90>

0.8 (c) Observation

1 iteration = linear regression on 1 batch
when batch size is small it takes more iteration to converge as compared to large batch size
$No.\ of\ iterations \propto \frac{1}{batch\ size}$

It converges fast when batch size is 10000 and takes too long to converge when batch size is 1000000.

0.9 Error on test data

0.9.1 Cost when batch size = 1

1.004445895587165

[36]:
```
cost1 = cost(X_test, res1, Y_test)
# print(cost1)
```

0.9.2 Cost when batch size = 100

0.9834276129619844

[39]:
```
cost2 = cost(X_test, res2, Y_test)
# print(cost2)
```

0.9.3 Cost when batch size = 10000

0.9863700639920684

[40]:
```
cost3 = cost(X_test, res3, Y_test)
# print(cost3)
```

0.9.4 Cost when batch size = 1000000

1.0180463660973345

[35]:
```
cost4 = cost(X_test, res4, Y_test)
# print(cost4)
```

```
1.0180463660973345
```

0.10 (e) Observation on plot of theta

When batch size is too small (r = 1) then plot is noisy as it update parameters based on only one example, as we increase batch size then plot becomes more smoother because parameters are updated based on some subset of data which gives enough information about data.

Question 3

0.1 Logistic Regression

[1]:
```
import seaborn as sb
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import threading
import time
from matplotlib import cm
# %matplotlib notebook

with open('data/q3/logisticX.csv') as fp:
    x=[]
    for line in fp:
        line=line[:-1]
        [x1,x2]=line.split(',')
        x.append(float(x1))
        x.append(float(x2))

with open('data/q3/logisticY.csv') as fp:
    y=[]
    for line in fp:
        y.append(float(line[:-1]))


X = np.array(x).reshape(100,2)
Y = np.array(y).reshape(100,1)
```

[2]:
```
def normalization(X):
    X_mean = np.mean(X)
    X_var= np.sum((X-X_mean)**2)/len(X)
    X_std_dev= np.sqrt(X_var)
    X_norm = (X- X_mean)/X_std_dev
    return X_norm
```

```python
X1= X[:,0:1]
X2=X[:,1:2]

X1_norm = normalization(X1)
X2_norm= normalization(X2)

X_norm = np.append(X1_norm, X2_norm, axis = 1)
X_norm = np.append(np.ones((X1_norm.shape[0], 1)), X_norm,axis = 1)
```

[3]:
```python
def sigma(V):
    return np.array([1/(1+np.exp(-i)) for i in V])



def gradient(V):
    t= Y-(sigma(V).reshape(V.shape[0],1))
    return np.dot(X_norm.transpose(), t)

def Hessian(Th):
#     t= np.dot(X_norm,theta)
    temp=sigma(np.dot(X_norm,Th))
    D= temp*(np.ones((temp.shape[0],1))-temp)
    t=np.dot(X_norm.T,np.diag(-D[:,0]))
    return (np.dot(t, X_norm))



def grad_des(X_norm):
    theta=np.zeros((3,1))
    for i in range(7):
        G = gradient(np.dot(X_norm,theta))
        H=Hessian(theta)
        H_inv= np.linalg.inv(H)
        theta= theta - np.dot(H_inv, G)
    return theta

def plot_data(X_norm, Y0, Y1, res):
    x10= Y0[:,0:1]
    x11=Y0[:,1:2]
    x00= Y1[:,0:1]
    x01=Y1[:,1:2]
    sb.set()
#     sb.set_style("ticks")
    fig, ax = plt.subplots()
    ax.scatter(x10,x11, marker="^", color='green', label= "1")
    ax.scatter(x00,x01, marker="P", color='red', label ="0")
    ax.legend()
    plt.xlabel('x1')
    plt.ylabel('x2')
```

```
    plt.title('x-y')
    Ydash = -(np.dot(X_norm[:,0:2],res[0:2,:]))/res[2][0]
    plt.plot(X_norm[:,1:2],Ydash,color='black')
    plt.show()
```

0.2 (a) Parameters Learned

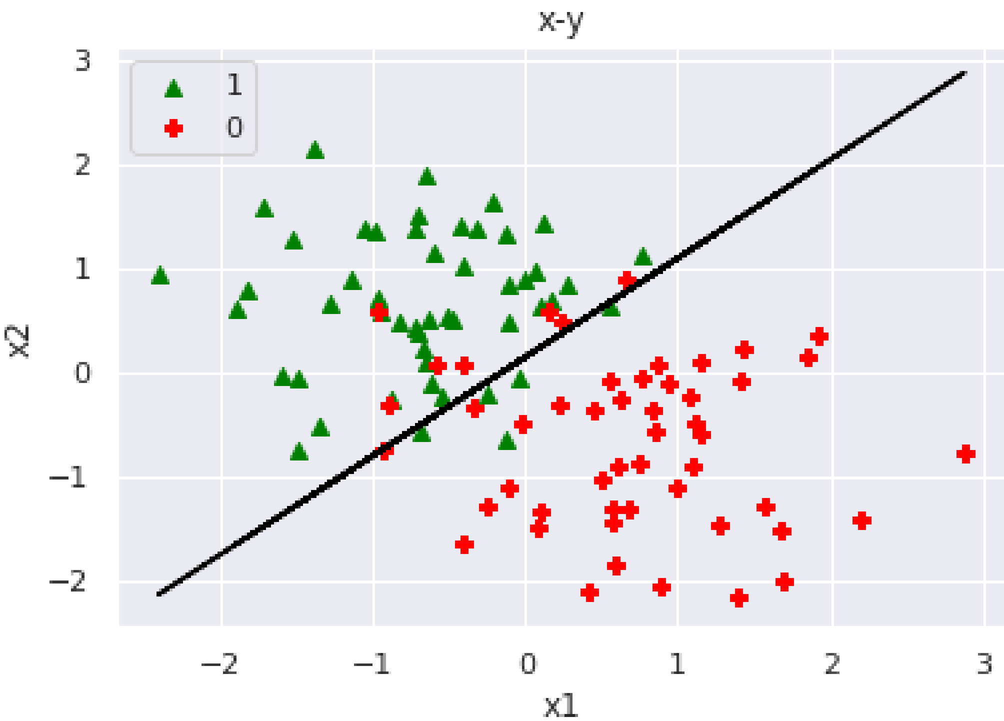
$\theta_0 = 0.40125316$
$\theta_1 = 2.5885477$
$\theta_2 = -2.72558849$

[4]:
```
res = grad_des(X_norm)
print(res)
Y0 =np.array([[X_norm[i][1],X_norm[i][2]]  for i in range(len(X_norm)) if␣
 ↪Y[i]==0])
Y1 =np.array([[X_norm[i][1],X_norm[i][2]]  for i in range(len(X_norm)) if␣
 ↪Y[i]==1])
```

```
[[ 0.40125316]
 [ 2.5885477 ]
 [-2.72558849]]
```

0.3 (b) Decision Boundary

[5]:
```
plot_data(X_norm, Y0, Y1, res)
```


x-y
1
0
x1
x2

Question 4

0.1 Gaussian Discrmimant Analysis

[1]:
```
import seaborn as sb
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
import threading
import time
from matplotlib import cm


with open('data/q4/q4x.dat') as fp:
    x=[]
    for line in fp:
        line=line[:-1]
        [x1,x2]=line.split('  ')
        x.append(float(x1))
        x.append(float(x2))

with open('data/q4/q4y.dat') as fp:
    y=[]
    for line in fp:
        y.append((line[:-1]))


X = np.array(x).reshape(100,2)
Y = np.array(y).reshape(100,1)

def normalization(X):
    X_mean = np.mean(X)
    X_var= np.sum((X-X_mean)**2)/len(X)
    X_std_dev= np.sqrt(X_var)
    X_norm = (X- X_mean)/X_std_dev
    return X_norm
```

[2]:

```python
def plot_data(X_alaska, X_canada):
    sb.set()
    fig, ax = plt.subplots(figsize=(8, 6), dpi= 80)
    color=["red:alaska", "blue:canada"]
    label1= ("alaska", "canada")
    ax.scatter(X_alaska[:,0],X_alaska[:,1], marker="^", c="blue",␣
↪label="Alaska-0")
    ax.scatter(X_canada[:,0],X_canada[:,1], marker="*",c="red", label =␣
↪"Canada-1")
    ax.legend()
    plt.xlabel('x0')
    plt.ylabel('x1')
    plt.show()


def plot_linear(X_norm1,X_alaska, X_canada):
    theta0 = np.log(phi/(1-phi))+ (mu0.T@cov_inv@mu0- mu1.T@cov_inv@mu1)/2
    theta12 = cov_inv@(mu1-mu0)
    theta=np.append(theta0, theta12, axis=0)
    X_norm1 = np.append(np.ones((len(X_norm),1)), X_norm, axis=1  )
    Y_dash = -(np.dot(X_norm1[:,0:2],theta[0:2,:]))/theta[2][0]

    sb.set()
    fig, ax = plt.subplots(figsize=(8, 6), dpi= 80)
    color=["red:alaska", "blue:canada"]
    label1= ("alaska", "canada")
    ax.scatter(X_alaska[:,0],X_alaska[:,1], marker="^", c="blue",␣
↪label="Alaska-0")
    ax.scatter(X_canada[:,0],X_canada[:,1], marker="*",c="red", label =␣
↪"Canada-1")
    ax.legend()
    plt.xlabel('x0')
    plt.ylabel('x1')
    plt.plot(X_norm1[:,1:2],Y_dash,color='black')
    plt.show()

def compute_parameters(cov0, cov1):
    cov0_inv = np.linalg.inv(cov0)
    cov1_inv = np.linalg.inv(cov1)
    cov0_det_sq= np.sqrt(np.linalg.det(cov0))
    cov1_det_sq= np.sqrt(np.linalg.det(cov1))
    diff_cov= (cov0_inv - cov1_inv)/2
    th0 = np.log(phi/(1-phi)) + np.log(cov0_det_sq/cov1_det_sq) + (mu0.
↪T@cov0_inv@mu0- mu1.T@cov1_inv@mu1)/2
    th1 = diff_cov[0][0]
    th2 = diff_cov[0][1]+ diff_cov[1][0]
    th3 = diff_cov[1][1]
```

```
    th45 = cov1_inv@mu1 - cov0_inv@mu0
    th4 = th45[0][0]
    th5 = th45[1][0]
    return(th0, th1, th2, th3, th4,th5)

def sol_quad_eq(x0, cov0, cov1):
    (th0, th1, th2, th3, th4,th5)= compute_parameters(cov0, cov1)
    a= th3
    b= th2*x0+th5
    c= th0+th1*x0*x0 + th4*x0
    d = (b**2) - (4*a*c)
    sol1 = (-b-np.sqrt(d))/(2*a)
    sol2 = (-b+np.sqrt(d))/(2*a)
    return(sol1)

def plot_quadratic(X_alaska, X_canada, cov0, cov1):
    X_norm1 = np.append(np.ones((len(X_norm),1)), X_norm, axis=1  )
    Y_dash = -(np.dot(X_norm1[:,0:2],theta[0:2,:]))/theta[2][0]

    sb.set()
    fig, ax = plt.subplots(figsize=(8, 6), dpi= 80)
    color=["red:alaska", "blue:canada"]
    label1= ("alaska", "canada")
    ax.scatter(X_alaska[:,0],X_alaska[:,1], marker="^", c="blue",␣
 ↪label="Alaska-0")
    ax.scatter(X_canada[:,0],X_canada[:,1], marker="*",c="red", label =␣
 ↪"Canada-1")
    ax.legend()
    plt.xlabel('x0')
    plt.ylabel('x1')
    x_ran= np.linspace(-2,2, 100)
    x1=np.array([sol_quad_eq(i, cov0, cov1) for i in x_ran]).reshape(100,1)
    plt.plot(X_norm1[:,1:2],Y_dash,color='green')
    plt.plot(x_ran, x1,color='black')
    plt.show();
```

[3]:
```
X0= X[:,0:1]
X1=X[:,1:2]
X0_norm = normalization(X0)
X1_norm= normalization(X1)
X_norm = np.append(X0_norm, X1_norm, axis = 1)
X_alaska = np.array([[X_norm[i][0],X_norm[i][1]]  for i in range(len(X_norm)) if␣
 ↪Y[i]=="Alaska"])
X_canada = np.array([[X_norm[i][0],X_norm[i][1]]  for i in range(len(X_norm)) if␣
 ↪Y[i]=="Canada"])
mu0 = np.mean(X_alaska,axis=0)
```

```
mu1 = np.mean(X_canada, axis=0)
x0 = X_alaska-mu0
x1= X_canada-mu1
cov = (np.dot(x0.T, x0) + np.dot(x1.T, x1))/100
cov_inv= np.linalg.inv(cov)
phi = len(X_alaska)/(len(X_alaska)+ len(X_canada))
mu0= mu0.reshape(2,1)
mu1= mu1.reshape(2,1)
print('mu0-',mu0)
print('mu1-',mu1)
print('sigma-',cov)


theta0 = np.log(phi/(1-phi))+ (mu0.T@cov_inv@mu0- mu1.T@cov_inv@mu1)/2
theta12 = cov_inv@(mu1-mu0)
theta=np.append(theta0, theta12, axis=0)
# print(theta12)
```

```
mu0- [[-0.75529433]
 [ 0.68509431]]
mu1- [[ 0.75529433]
 [-0.68509431]]
sigma- [[ 0.42953048 -0.02247228]
 [-0.02247228  0.53064579]]
```

0.1.1 (a)

$\Sigma = \sum_{i=1}^{m} \frac{(x^{(i)}-\mu_y^{(i)})(x^{(i)}-\mu_y^{(i)})^T}{m}$

$\mu0 = \sum_{i=1}^{m} \frac{I\{y^{(i)}=0\}x^{(i)}}{\sum i=1^m I\{y^{(i)}=0\}}$

$\mu1 = \sum_{i=1}^{m} \frac{I\{y^{(i)}=1\}x^{(i)}}{\sum i=1^m I\{y^{(i)}=1\}}$

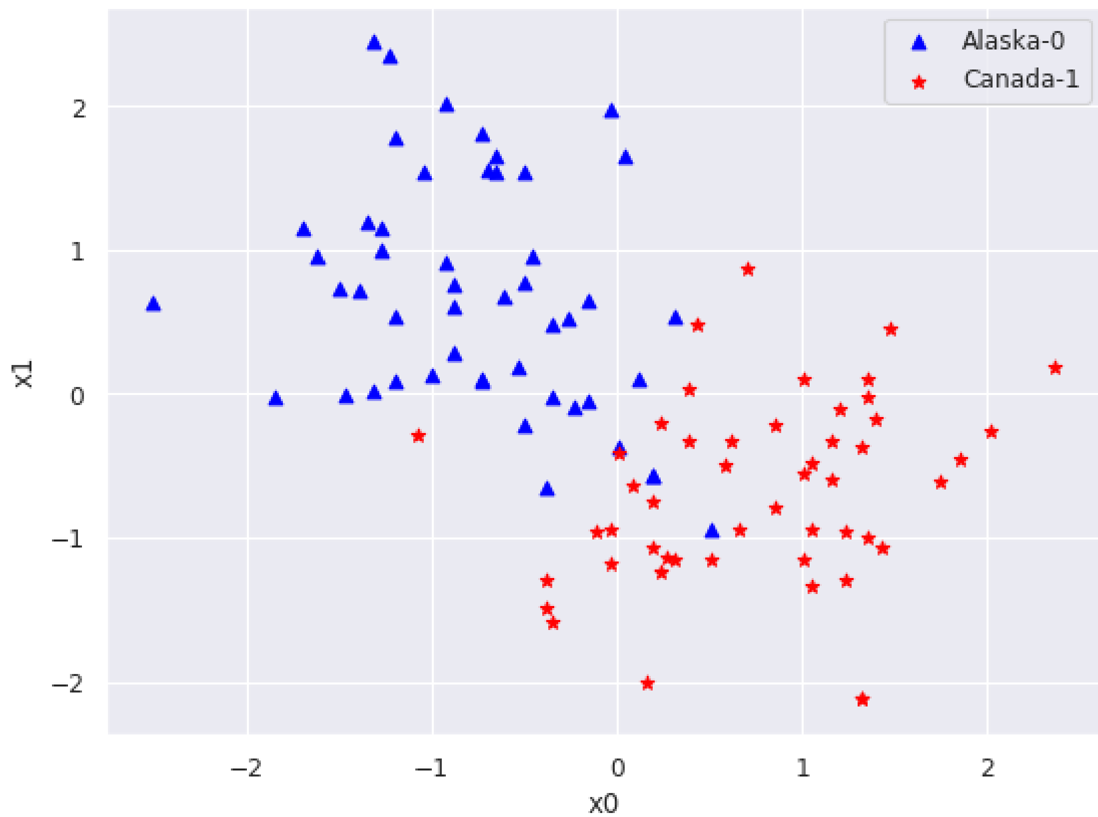
$\mu_0 = [[-0.75529433], [0.68509431]]$

$\mu_1 = [[0.75529433], [-0.68509431]]$

$\Sigma = [[0.42953048, -0.02247228],$
$[-0.02247228, 0.53064579]]$

0.2 (b) Data

[4]:
```
plot_data(X_alaska, X_canada)
```



0.3 (c)Linear Boundary

0.3.1 Equation of decision boundary when $\Sigma_0 = \Sigma_1 = \Sigma$

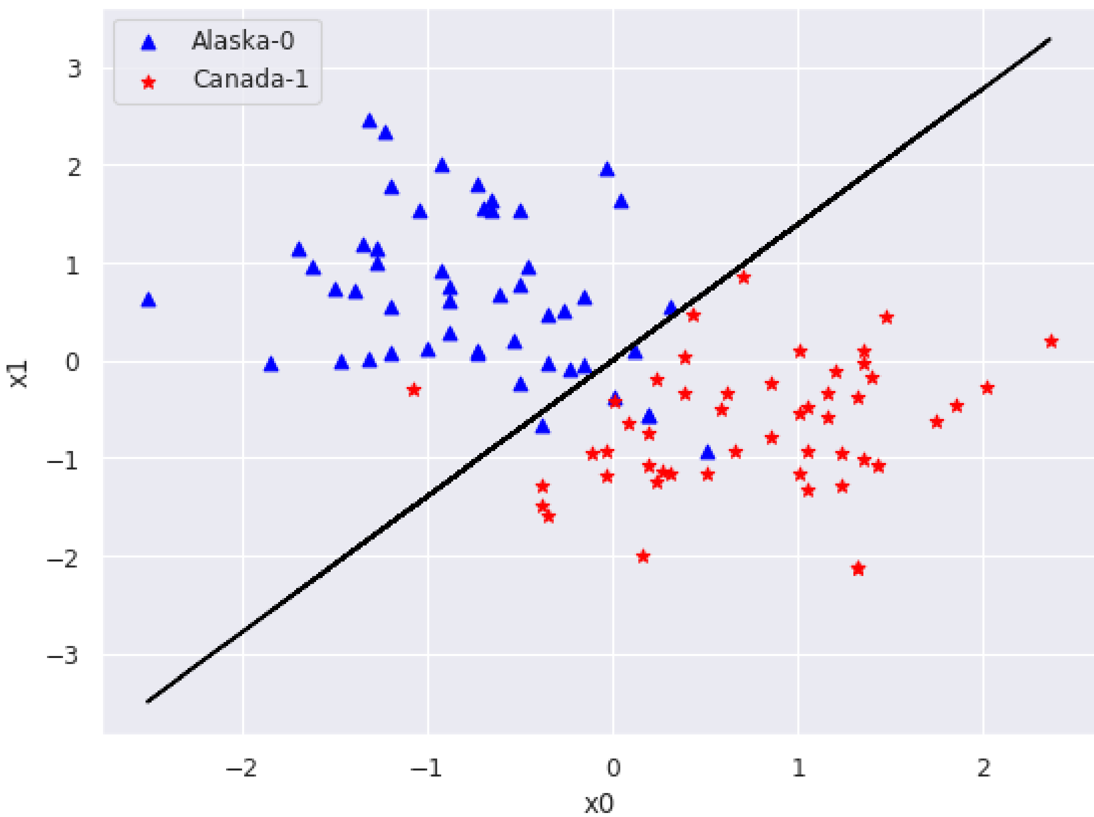
$\Sigma = \sum_{i=1}^{m} \frac{(x^{(i)} - \mu_y^{(i)})(x^{(i)} - \mu_y^{(i)})^T}{m}$

$\log(\frac{\phi}{(1-\phi)}) + x^T(\Sigma^{-1}\mu_1 - \Sigma^{-1}\mu_0) + \frac{(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1)}{2} = 0$

$\theta\_0 = \log(\frac{\phi}{(1-\phi)} + \frac{(\mu_0\Sigma^{-1}\mu_0 - \mu_1\Sigma^{-1}\mu_1)}{2}$

$[\theta_1, \theta_2] = \Sigma^{-1}\mu_1 - \Sigma^{-1}\mu_0$

[5]:
```
plot_linear(X_norm,X_alaska, X_canada)
```



0.4 (d)

$\$\ \Sigma\_0 = [[0.38158978 :,: -0.15486516] : \$$
$\$\qquad [-0.15486516\,,\, 0.64773717]]\$$

$\Sigma_1 = [[0.47747117\,,\, 0.1099206]$

$\qquad [0.1099206\,,\, 0.41355441]]$

$\mu_0 = [[-0.75529433\,,\, [0.68509431]]$

$\mu_1 = [[0.75529433]\,,\, [-0.68509431]]$

[6]:
```
cov0= ((X_alaska-mu0.T).T@(X_alaska-mu0.T))/50
cov1= ((X_canada-mu1.T).T@(X_canada-mu1.T))/50
print('sigma0-',cov0)
print('sigma1-',cov1)
```

```
sigma0- [[ 0.38158978 -0.15486516]
 [-0.15486516  0.64773717]]
sigma1- [[0.47747117 0.1099206 ]
 [0.1099206  0.41355441]]
```

0.5 (e)Quadratic Boundary

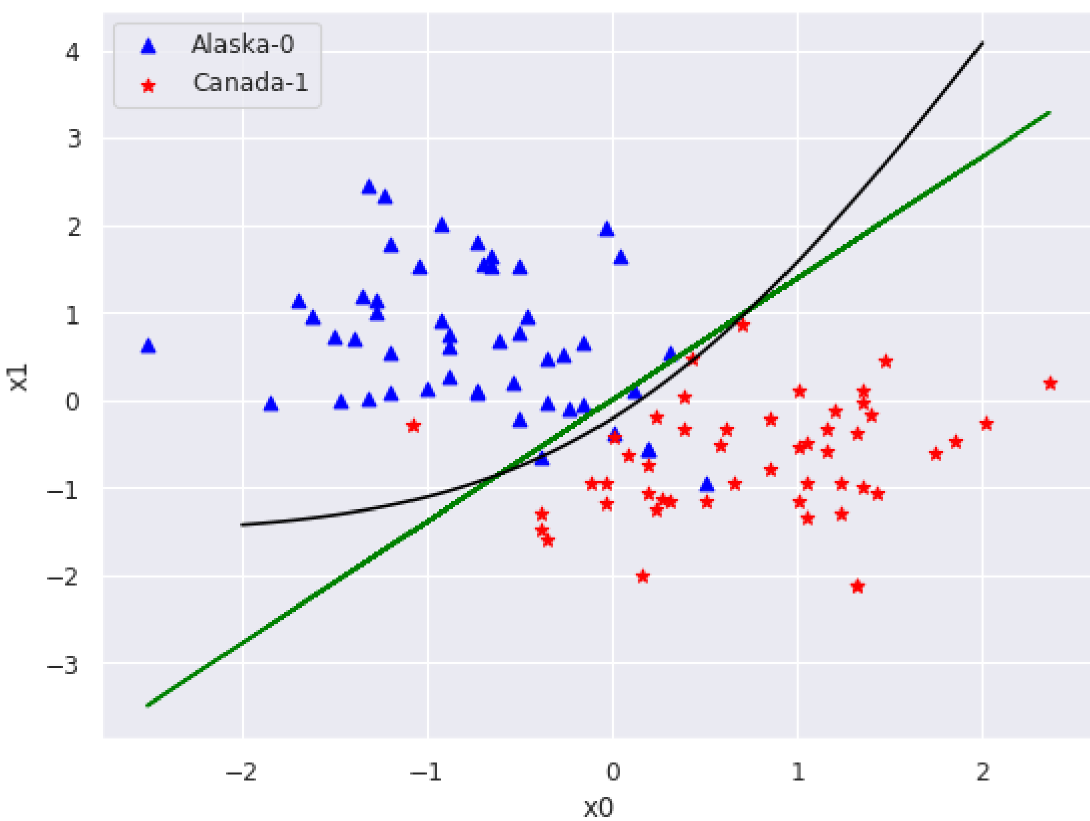
$\log(\frac{\phi}{(1-\phi)}) + \frac{1}{2}\log(\frac{|\Sigma_1|}{|\Sigma_0|} + x^T \frac{(\Sigma_0^{-1} - \Sigma_1^{-1})}{2} x + x^T(\Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0) + \frac{\mu_0^T \Sigma_0^{-1}\mu_0 - \mu_1^T \Sigma_1^{-1}\mu_1}{2} = 0$

$\theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_1 x_2 + \theta_4 x_1 + \theta_5 x_2 = 0$

$\theta_0 = \log(\frac{\phi}{(1-\phi)}) + \frac{1}{2}\log(\frac{|\Sigma_1^{-1}|}{|\Sigma_0^{-1}|} + \frac{\mu_0^T \Sigma_0^{-1}\mu_0 - \mu_1^T \Sigma_1^{-1}\mu_1}{2}$

let $A = \frac{(\Sigma_0^{-1} - \Sigma_1^{-1})}{2}$
$\theta_1 = A_{00}$
$\theta_2 = A_{11}$
$\theta_3 = A_{01} + A_{10}$
$[\theta\_4, \theta\_5] = \Sigma_1^{-1}\mu_1 - \Sigma_0^{-1}\mu_0$

[7]:
```
plot_quadratic(X_alaska, X_canada,cov0, cov1)
```



0.6 (f)Observation

As from observation on plot Quadratic boundary is better classifyingthe examples then linear boundary. In linear boundary 4 0's are misclasified as 1's andin quadratic only 2 0's are misclassified as 1′s.