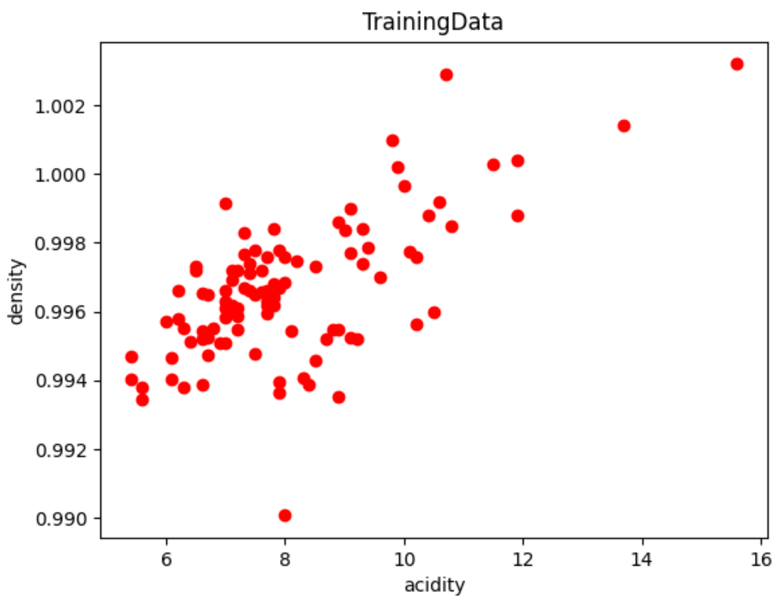
COL774 - Machine Learning
CSE, IIT Delhi (SEM1, 2020-21)

Assignment 1

2018CS10388
Sharique Shamim

1 Linear Regression

In this part we were supposed to implement Linear Regression using Batch Gradient Descent. First I read the data from linearX.csv and linearY.csv into trainX and trainY respectively. Then I normalized the data to set mean of the data to zero and variance to one. To create the design matrix X(m x (n + 1)), append column of ones to trainX, and reshape trainY to Y(m x 1) matrix.



Hypothesis: $h_\theta(x) = \Theta^T \cdot X$

Cost Function: $J(\Theta) = \frac{1}{2}\sum_{i=1}^{m}\{y^{(i)} - h_\theta(x)^{(i)}\}^2$

LMS Algorithm:

$\Theta_j = \Theta_j$ - $\eta \cdot \frac{\partial J(\Theta)}{\partial \Theta_j}$

$\frac{\partial J(\Theta)}{\partial \Theta_j} = -\sum_{i=1}^{m}(y^{(i)} - h_\theta(x)^{(i)}) \cdot x^{(i)}$

$\Theta_j = \Theta_j + \eta \cdot \sum_{i=1}^{m}(y^{(i)} - h_\theta(x)^{(i)}) \cdot x^{(i)}$

Algorithm:

Initialize Θ((n + 1) x 1) $\rightarrow$ zeroes

while not converged:

$\Theta = \Theta + \eta \cdot X^T \cdot (Y - h)$

return Θ

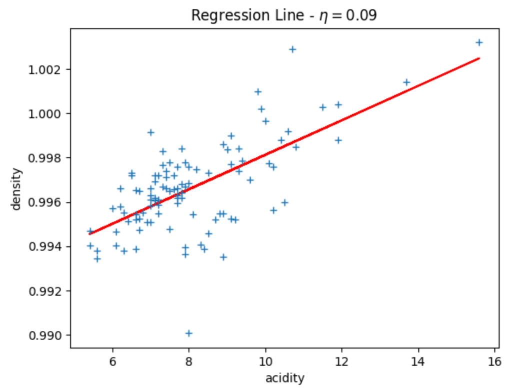
(a) **Observations**:

Learning Rate(η): 0.09

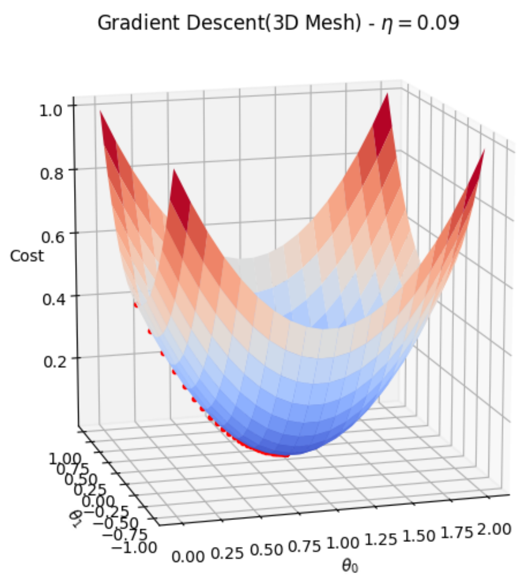
Stopping Criteria: When $(J^t(\theta) - J^{t-1}(\theta)) < 10^{-20}$ or iterations = 20000

Final Parameters: [0.99662003 , 0.0013402]

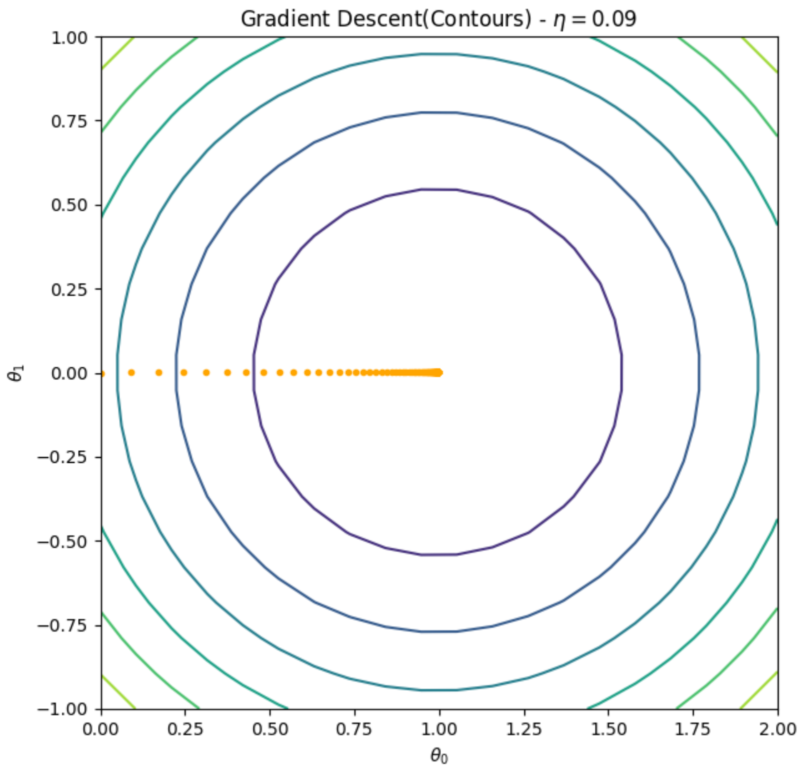
(b) **Best fitting line for the training data**:



(c) **3D Mesh**



(d) **Contour Plot**



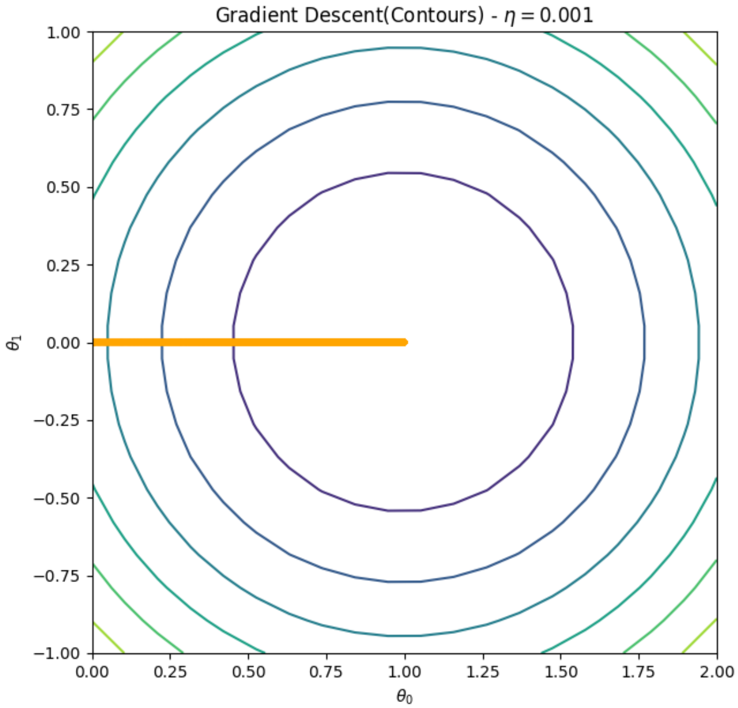

```
Learning Rate = 0.09
iteration 100: error = 6.629818354221584e-10 cost = 1.1979836170215754e-06 theta = [0.99654018],[0.00134009]
iteration 200: error = 4.266928471793538e-18 cost = 1.1947898110041953e-06 theta = [0.99662009],[0.0013402]
Max Iterations =  231
Final Cost = 1.1947898109837215e-06
Final Parameters = [0.9966201],[0.0013402]
```

Max Iterations = 231

Final Cost = 1.1947898109837215e-06

Final Parameters = [0.9966201],[0.0013402]

(e) **Contour Plot for** $\eta = 0.001$



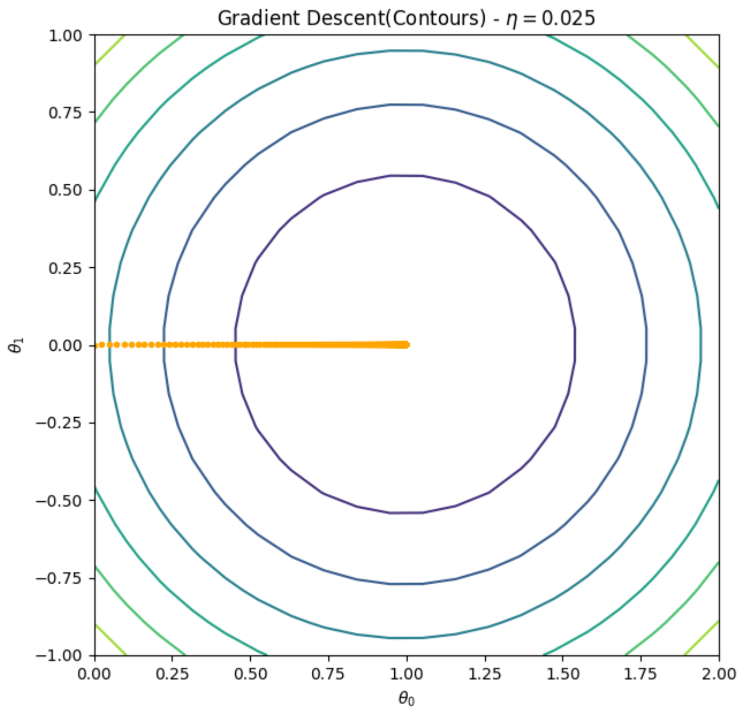

```
Learning Rate = 0.001
iteration 1000: error = 0.0001344894888585385 cost = 0.0671450889220348 theta = [0.63016745],[0.00084741]
iteration 2000: error = 1.818296886041751e-05 cost = 0.009079044267385353 theta = [0.86187714],[0.001159]
iteration 3000: error = 2.458336033432087e-06 cost = 0.0012285193619488448 theta = [0.94707573],[0.00127357]
iteration 4000: error = 3.3236684832292835e-07 cost = 0.00016712898040253016 theta = [0.97840286],[0.0013157]
iteration 5000: error = 4.493597309773758e-08 cost = 2.362907999985916e-05 theta = [0.9899217],[0.00133119]
iteration 6000: error = 6.075340210556875e-09 cost = 4.2279041708651665e-06 theta = [0.99415713],[0.00133688]
iteration 7000: error = 8.213855432378291e-10 cost = 1.6048666461609776e-06 theta = [0.99571448],[0.00133898]
iteration 8000: error = 1.1105126415481466e-10 cost = 1.250232168507431e-06 theta = [0.99628711],[0.00133975]
iteration 9000: error = 1.5014122625212856e-11 cost = 1.2022856135847292e-06 theta = [0.99649766],[0.00134003]
iteration 10000: error = 2.02990826586057e-12 cost = 1.1958032429361145e-06 theta = [0.99657508],[0.00134014]
iteration 11000: error = 2.7444345641138056e-13 cost = 1.1949268269080184e-06 theta = [0.99660355],[0.00134017]
iteration 12000: error = 3.710472781721138e-14 cost = 1.1948083355264065e-06 theta = [0.99661401],[0.00134019]
iteration 13000: error = 5.016555806487146e-15 cost = 1.194792315500449e-06 theta = [0.99661786],[0.00134019]
iteration 14000: error = 6.782355862507529e-16 cost = 1.1947901495941489e-06 theta = [0.99661928],[0.00134019]
iteration 15000: error = 9.169661049619835e-17 cost = 1.1947898567637768e-06 theta = [0.9966198],[0.0013402]
iteration 16000: error = 1.2382774655910617e-17 cost = 1.1947898171731335e-06 theta = [0.99661999],[0.0013402]
iteration 17000: error = 1.689407213298702e-18 cost = 1.194789811820469e-06 theta = [0.99662006],[0.0013402]
iteration 18000: error = 2.2742834633777964e-19 cost = 1.1947898110967934e-06 theta = [0.99662008],[0.0013402]
Max Iterations =  18972
Final Cost = 1.1947898109998543e-06
Final Parameters = [0.99662009],[0.0013402]
```

Max Iterations = 18972

Final Cost = 1.1947898109998543e-06

Final Parameters = [0.99662009],[0.0013402]

(f) **Contour Plot for $\eta = 0.025$**



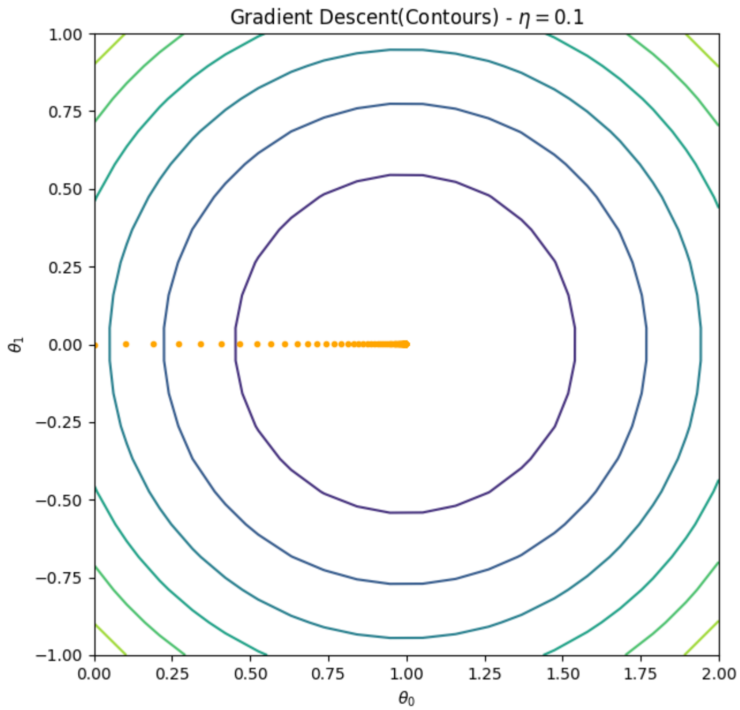

```
Learning Rate = 0.025
iteration 100: error = 0.00016309892190016556 cost = 0.0031413651722180043 theta = [0.91737157],[0.00123363]
iteration 200: error = 1.0312743831904957e-06 cost = 2.1050085213926034e-05 theta = [0.99031847],[0.00133172]
iteration 300: error = 6.520747292675762e-09 cost = 1.3203348316445429e-06 theta = [0.99611901],[0.00133952]
iteration 400: error = 4.1230681123605336e-11 cost = 1.1955836320723465e-06 theta = [0.99658025],[0.00134014]
iteration 500: error = 2.6070156593434887e-13 cost = 1.1947948303139187e-06 theta = [0.99661693],[0.00134019]
iteration 600: error = 1.6484137832669093e-15 cost = 1.19478984272088e-06 theta = [0.99661985],[0.0013402]
iteration 700: error = 1.0422740415964166e-17 cost = 1.1947898111843374e-06 theta = [0.99662008],[0.0013402]
iteration 800: error = 6.352747104407253e-20 cost = 1.194789810984934e-06 theta = [0.9966201],[0.0013402]
Max Iterations =  824
Final Cost = 1.1947898109840465e-06
Final Parameters = [0.9966201],[0.0013402]
```

Max Iterations = 824

Final Cost = 1.1947898109840465e-06

Final Parameters = [0.9966201],[0.0013402]

(g) **Contour Plot for $\eta = 0.1$**




```
Learning Rate = 0.1
iteration 100: error = 8.218651085846134e-11 cost = 1.1951401850562593e-06 theta = [0.99659363],[0.00134016]
iteration 200: error = 5.505714157152952e-20 cost = 1.1947898109839089e-06 theta = [0.9966201],[0.0013402]
Max Iterations =  208
Final Cost = 1.1947898109837147e-06
Final Parameters = [0.9966201],[0.0013402]
```

Max Iterations = 208

Final Cost = 1.1947898109837147e-06

Final Parameters = [0.9966201],[0.0013402]

Conclusion: As the learning rate increases the number of iterations taken to converge decreases.

2 Sampling and Stochastic Gradient Descent

In this part we were supposed to sample data and implement Stochastic Gradient Descent.

(a) **Sampling**:

$X_1 \sim \mathcal{N}(3, 4) \qquad X_2 \sim \mathcal{N}(-1, 4) \qquad noise \sim \mathcal{N}(0, 2)$

X = [ones, X_1, X_2]

θ = [3., 1., 2.]

Y = $\theta^T \cdot X +$ noise

(b) **Stochastic Gradient Descent**:

Initialize Θ((n + 1) x 1) $\rightarrow$ zeroes

while not converged:

for i in range(m/r):

x = $X[i * r : (i + 1) * r,]$

y = $Y[i * r : (i + 1) * r,]$

h = $x \cdot \theta$

$\theta = \theta + \eta \cdot x^T \cdot (y - h)$

Convergence Criteria:

if epoch > 7 and $(J_b^t(\theta) - J_b^{t-1}(\theta)) < 10^{-7}$ or iterations > 25000 **then** converged

(c) **Observations**: Max Iterations for different Batch Sizes

r = 1, 100, 10000, 1000000

```
Batch Size (r) = 1
iteration 1: error = 0.5029794109282136 w = [3.00920305],[0.96389771],[2.00218106]
iteration 2: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
iteration 3: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
iteration 4: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
iteration 5: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
iteration 6: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
iteration 7: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
iteration 8: error = 0.0 w = [3.00920305],[0.96389771],[2.00218106]
```

```
Batch Size (r) = 100
iteration 1: error = 1.0203129559549962 w = [2.81164362],[1.03761322],[1.98730319]
iteration 2: error = 0.0032528892153365074 w = [2.98354481],[0.99994356],[1.99989438]
iteration 3: error = 0.0005285175913627427 w = [2.99493333],[0.99744794],[2.00072855]
iteration 4: error = 3.6388341589033146e-05 w = [2.99568782],[0.9972826],[2.00078381]
iteration 5: error = 2.416771651247984e-06 w = [2.99573781],[0.99727165],[2.00078748]
iteration 6: error = 1.601385206662087e-07 w = [2.99574112],[0.99727092],[2.00078772]
iteration 7: error = 1.060935539420882e-08 w = [2.99574134],[0.99727087],[2.00078773]
iteration 8: error = 7.028744253290142e-10 w = [2.99574135],[0.99727087],[2.00078774]
```

```
Batch Size (r) = 10000
iteration 10: error = 0.03932605135277001 w = [0.88172528],[1.45010136],[1.80763754]
iteration 20: error = 0.020594615780346714 w = [1.38595558],[1.35234721],[1.88152759]
iteration 30: error = 0.011931227726361238 w = [1.77034081],[1.26859721],[1.91014447]
iteration 40: error = 0.006904802563734158 w = [2.06336602],[1.2045861],[1.93145935]
iteration 50: error = 0.003990608636809556 w = [2.28674551],[1.15578603],[1.94769912]
iteration 60: error = 0.0023023010849012593 w = [2.45703255],[1.11858463],[1.96007888]
iteration 70: error = 0.0013251554218607353 w = [2.58684607],[1.09022518],[1.96951624]
iteration 80: error = 0.0007603425717348511 w = [2.68580573],[1.06860616],[1.97671055]
iteration 90: error = 0.0004344279439055798 w = [2.76124484],[1.05212551],[1.98219492]
iteration 100: error = 0.0002467944648293363 w = [2.81875371],[1.03956196],[1.98637578]
iteration 110: error = 0.00013910100932679192 w = [2.86259397],[1.02998449],[1.98956294]
iteration 120: error = 7.754334468634827e-05 w = [2.89601436],[1.02268337],[1.99199258]
iteration 130: error = 4.25527501735079e-05 w = [2.92149145],[1.01711757],[1.99384475]
iteration 140: error = 2.2815168060685842e-05 w = [2.94091318],[1.01287464],[1.9952567]
iteration 150: error = 1.179982430177251e-05 w = [2.9557188],[1.00964016],[1.99633306]
iteration 160: error = 5.745166904547183e-06 w = [2.96700544],[1.00717445],[1.99715359]
iteration 170: error = 2.490923071363227e-06 w = [2.97560949],[1.00529478],[1.9977791]
iteration 180: error = 8.012714466376636e-07 w = [2.98216854],[1.00386187],[1.99825594]
```

```
Batch Size (r) = 1000000
iteration 1000: error = 0.0003752636927039088 w = [0.88177938],[1.45007354],[1.80772352]
iteration 2000: error = 0.00020297873138419575 w = [1.3859972],[1.35233528],[1.88162143]
iteration 3000: error = 0.00011795304437955956 w = [1.77037232],[1.26859712],[1.91024402]
iteration 4000: error = 6.854659595956214e-05 w = [2.06338986],[1.20459502],[1.93156326]
iteration 5000: error = 3.983479996239048e-05 w = [2.28676353],[1.15580181],[1.94780634]
iteration 6000: error = 2.314938131897648e-05 w = [2.45704616],[1.11860563],[1.96018863]
iteration 7000: error = 1.3452906904021233e-05 w = [2.58685633],[1.09025016],[1.96962792]
iteration 8000: error = 7.817949935651214e-06 w = [2.68581347],[1.06863417],[1.9768237]
iteration 9000: error = 4.54328136090254e-06 w = [2.76125065],[1.05215583],[1.9823092]
iteration 10000: error = 2.6402580852735724e-06 w = [2.81875807],[1.03959403],[1.98649092]
iteration 11000: error = 1.534345377418589e-06 w = [2.86259723],[1.03001789],[1.98967873]
iteration 12000: error = 8.916612175280392e-07 w = [2.89601678],[1.0227178],[1.99210887]
iteration 13000: error = 5.181752020799735e-07 w = [2.92149323],[1.01715277],[1.99396142]
iteration 14000: error = 3.0112954885197496e-07 w = [2.94091449],[1.01291043],[1.99537366]
iteration 15000: error = 1.7499680526888994e-07 w = [2.95571974],[1.00967641],[1.99645024]
iteration 16000: error = 1.0169670283666221e-07 w = [2.96700611],[1.00721104],[1.99727094]
```

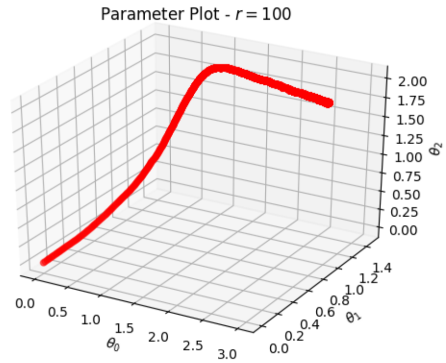
Conclusion: Number of Iterations increases as the Batch Size increases.

Test Error on q2test:

1. For r = 1 : Training Error = 0.6731270320142997, Test Error = 0.9958435765081824
2. For r = 100 : Training Error = 0.8019260284240661, Test Error = 0.9833262043926705
3. For r = 10000 : Training Error = 1.0017006182677617, Test Error = 0.983166852127419
4. For r = 1000000 : Training Error = 0.9997146049993177, Test Error = 0.9866873158769277

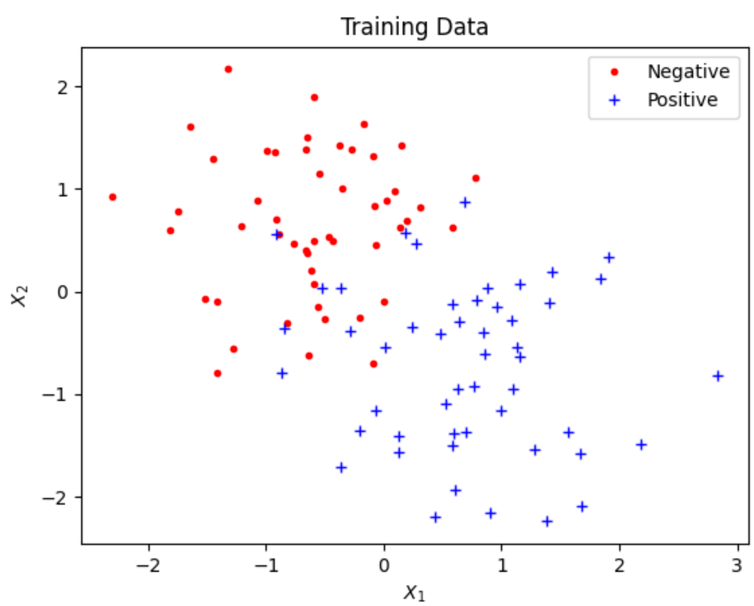
Test Error decreases with increasing batch size.

(d) **3D Parameter Plot**: At first θ changes steeply then gradually. For smaller batch size their is noise in the movement but for larger batch sizes the movement is smooth.



3 Logistic Regression

In this part we were supposed to implement Logistic Regresion using Newton's Method. First I read the data from logisticX.csv and logisticY.csv into trainX and trainY respectively. Then I normalized the data to set mean of the data to zero and variance to one. To create the design matrix X(m x (n + 1)), append column of ones to trainX, and reshape trainY to Y(m x 1) matrix.



Hypothesis: $h_\theta(x) = g(\Theta^T \cdot X) = \frac{1}{1+e^{\theta^T \cdot X}}$

Log Likelihood: $l(\Theta) = \sum_{i=1}^{m}\{y^{(i)} \cdot log(h(x^{(i)})) + (1 - y^{(i)}) \cdot log(1 - h_\theta(x)^{(i)}\}$

Newton's Method:

$\Theta := \Theta - \frac{l'(\theta)}{l"(\theta)}$

Algorithm:

Initialize Θ((n + 1) x 1) $\rightarrow$ zeroes

while not converged:

$\Theta = \Theta - H^{-1} \cdot dJ(\Theta)$

return Θ

(a) **Observations**:

Initial Paramaters: [0. 0. 0.]

Stopping Criteria: if $J^{(t)}(\theta) - J^{(t-1)}(\theta) < 10^{-10}$ then converged

Final Parameters: [0.40125316, 2.5885477, -2.72558849]

(b) **Decision Boundary**

4 Gaussian Discriminant Analysis

In this part we were supposed to implement Gaussian Discriminant. First I read the data from q4x.dat and q4y.dat into trainX and trainY respectively. Then I normalized the data to set mean of the data to zero and variance to one. Y is an array of strings so I mapped it to binary values 0 and 1. Class0 refers to Alaska and Class1 refers to Canada

GDA Parameters:

$$\phi = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 0\} \cdot x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 1\} \cdot x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 1\}}$$

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

$$\sum_0 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 0\} \cdot (x^{(i)} - \mu_y^{(i)})(x^{(i)} - \mu_y^{(i)})^T}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 0\}}$$

$$\sum_1 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 1\}(x^{(i)} - \mu_y^{(i)})(x^{(i)} - \mu_y^{(i)})^T}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} == 1\}}$$

(a) **Observations**:

$$\phi = 0.5$$

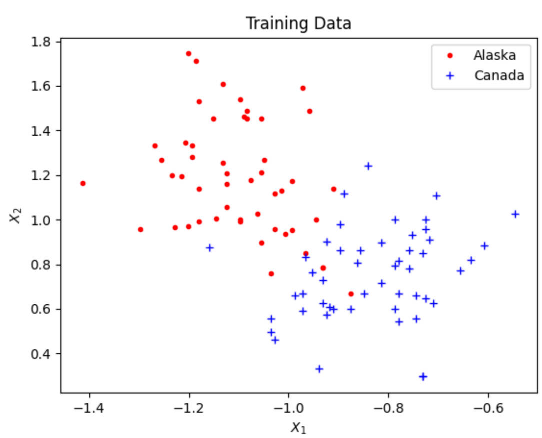
$$\mu_0 = \begin{pmatrix} -1.10106488 & 1.18368785 \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} -0.8315402 & 0.74891723 \end{pmatrix}$$

$$\sum = \begin{pmatrix} 0.0136741 & -0.00127227 \\ -0.00127227 & 0.05342744 \end{pmatrix}$$

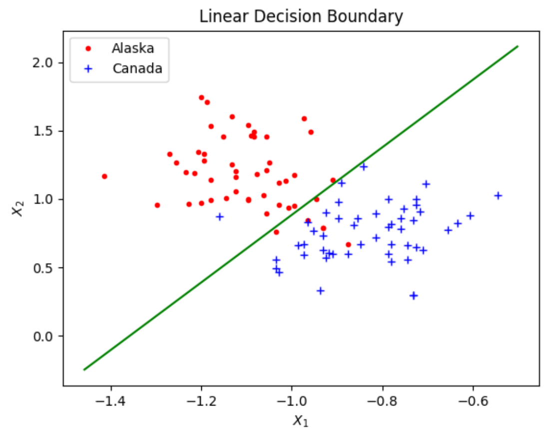
(b) **Training Data**:

$Y = 0$ if Alaska else $Y = 1$



(c) **Linear Decision Boundary**:

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



(d) **Observations**:

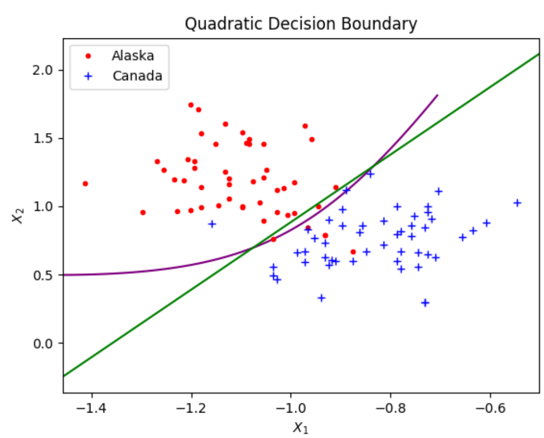
$\mu_0 = \begin{pmatrix} -1.10106488 & 1.18368785 \end{pmatrix}$

$\mu_1 = \begin{pmatrix} -0.8315402 & 0.74891723 \end{pmatrix}$

$\sum_0 = \begin{pmatrix} 0.0121479 & -0.0087677 \\ -0.0087677 & 0.06521665 \end{pmatrix}$

$\sum_1 = \begin{pmatrix} 0.01520029 & 0.00622316 \\ 0.00622316 & 0.04163824 \end{pmatrix}$

(e) **Quadratic Decision Boundary**



$x^T(\sum_1^{-1} - \sum_0^{-1})x \ + \ 2(\mu_0 \sum_0^{-1} - \mu_1 \sum_1^{-1})x \ + \ (\mu_1^T \sum_1^{-1} \mu_1 \ - \ \mu_0^T \sum_0^{-1} \mu_0) \ = \ log|\sum_0|$ - $log|\sum_1| \ +$

$$2log(\frac{\phi}{1-\phi})$$

(f) **Linear Boundary vs Quadratic Boundary**:

Linear Boundary almost passes through origin. It can be seen that both models have a good accuracy when it comes to classification. However the quadratic boundary is slightly more accurate by 2-3 examples and implies that the underlying distribution modelling the data is Gaussian.