COL774 ML Assignment - 1

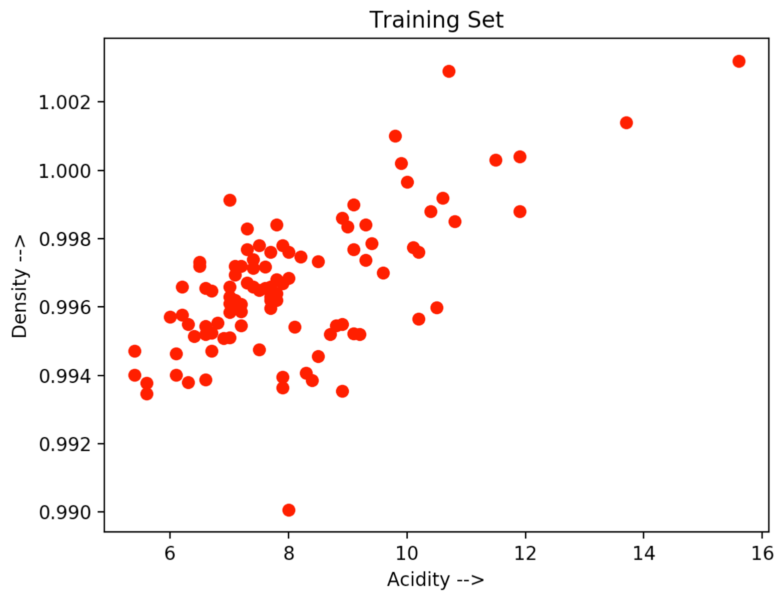
Ankur Sharma
2015CS50278

1. Linear Regression

In this part, we were supposed to implement linear regression using batch gradient descent. Here are a few key points about the implementation:
I have tried to use vectorised implementations as far as possible so as to optimise the speed of my code. Design Matrix(X) is first created by reading the data from the 'linearX.csv' file. Appending the column of ones at the start, we have our matrix X of dimensions (m) x (n + 1) ready with us. Similarly, the (n+1) dimensional vector 'y' of training examples has been created after reading the file 'linearY.csv'.

Note that the feature scaling has been done using mean normalisation so as to set the mean of the data to zero and the variance to one.

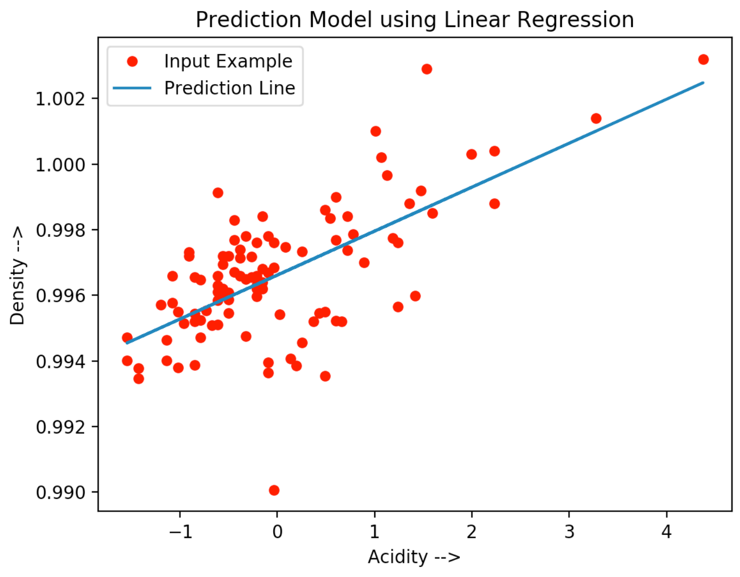
Once we have the matrix X and vector y, we zero-initialised our theta-vector and define a cost function using least-mean squared error that we have to minimise. Finally, we ran Gradient Descent Algorithm so as to get the optimal value of parameters theta.



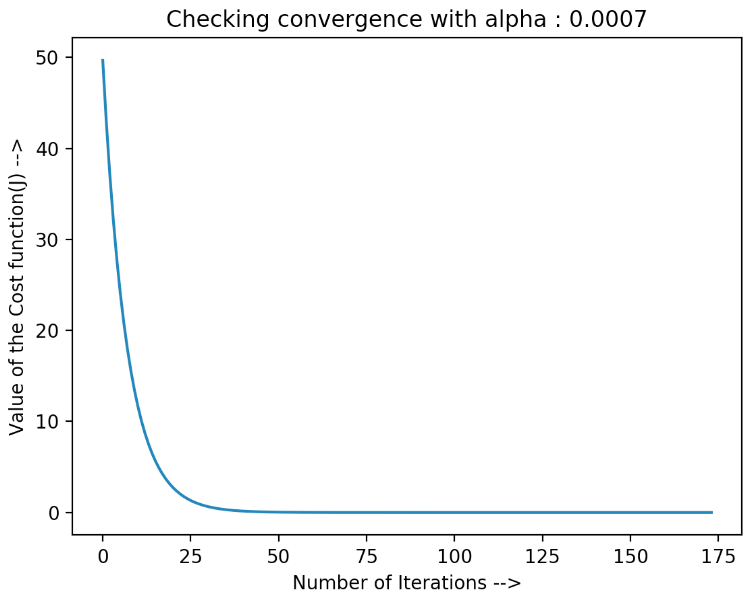
(a)

1. Theta vector was initialised to zero before running Batch Gradient Descent Algorithm.
   Initial Parameters = [0 0]'   Initial Value of Cost Function = 49.6627
2. Learning rate : 0.007
3. Convergence Check : If the absolute difference in the value of cost function in consecutive iterations becomes less than a certain epsilon(10^{-10}), then we declare that the algorithm converged.
4. At the end of GD, we have
   Final Parameters = [0.99662 0.00134]'
   Number of iterations taken to converge = 174
5. *J(theta) = 0.5*(X*theta - y)*^{T}*(X*theta - y)*
   At every valid iteration in GD until convergence, Theta update has been done as:
   *theta(t + 1) = theta(t) - eta * X*^{T}*(X*theta - y)*

(b) Best fitting line for the given training data can be seen from the blue lines as shown below.

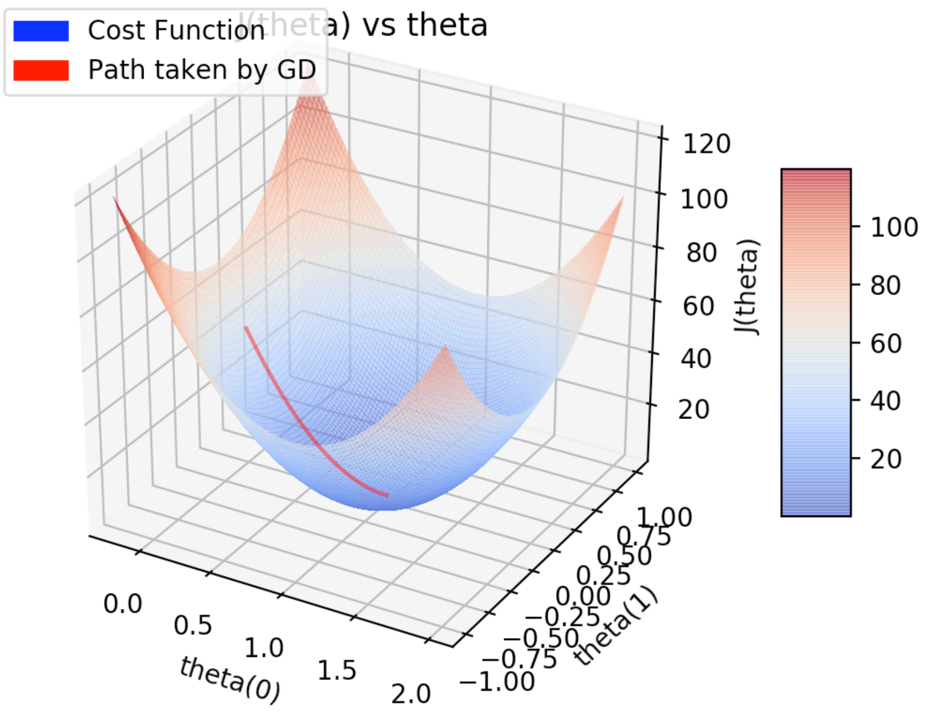


Blue line is our prediction model which can be used for generalising to new examples.



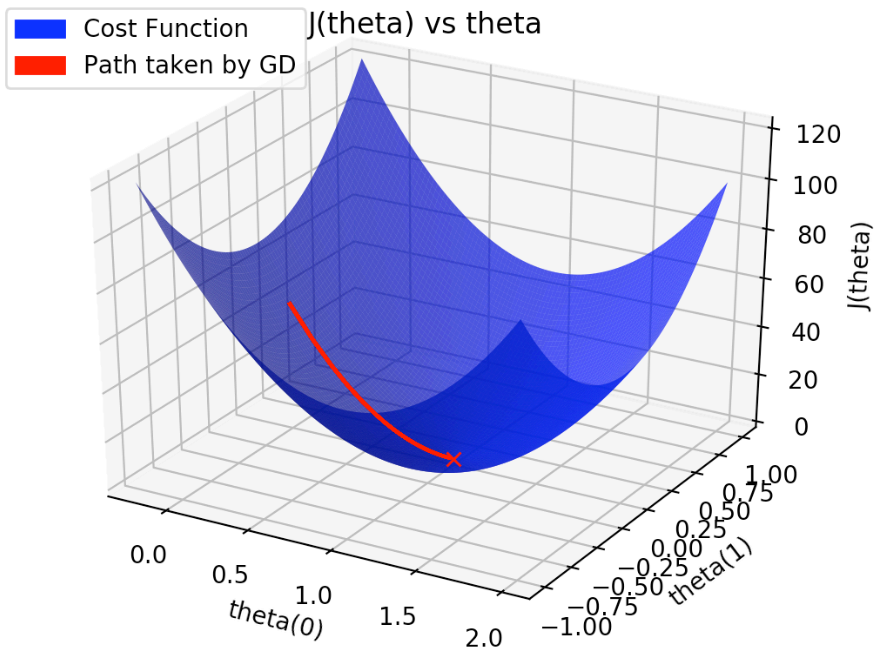
As we can see that the value of cost function decreases on every iteration and smoothly at this value of learning rate. Hence, we can say that the given value of eta(or alpha) is a good choice.

J(theta) should decrease on every iteration and the same has been confirmed from our observations.



(c) To visualise the cost-function in 3D, I sampled out 100 points along theta-0 and 100 points along theta-1 to form a 2D mesh grid with 10000 points in it. Evaluating the value of cost function at each of these points helps us to form a 3D surface using *plot_surface* method in matplotlib. The different values of cost function has also been shown using a colormap so that visualising can be done easier.

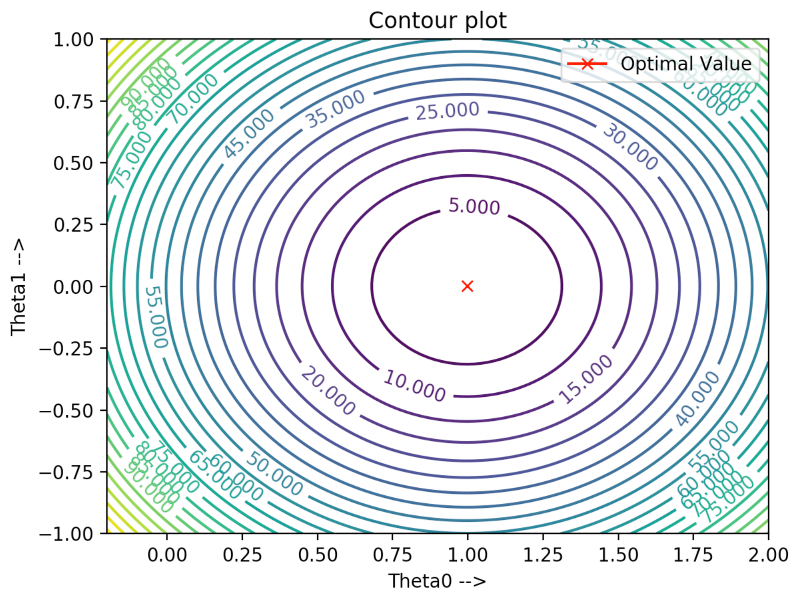
The path taken by the GD has been shown in red as can be seen.



To show the visualisation, I have used the 3D animation toolbox available in **matplotlib.animation** which allows us to view the gradient descent on 3D surface frame by frame.

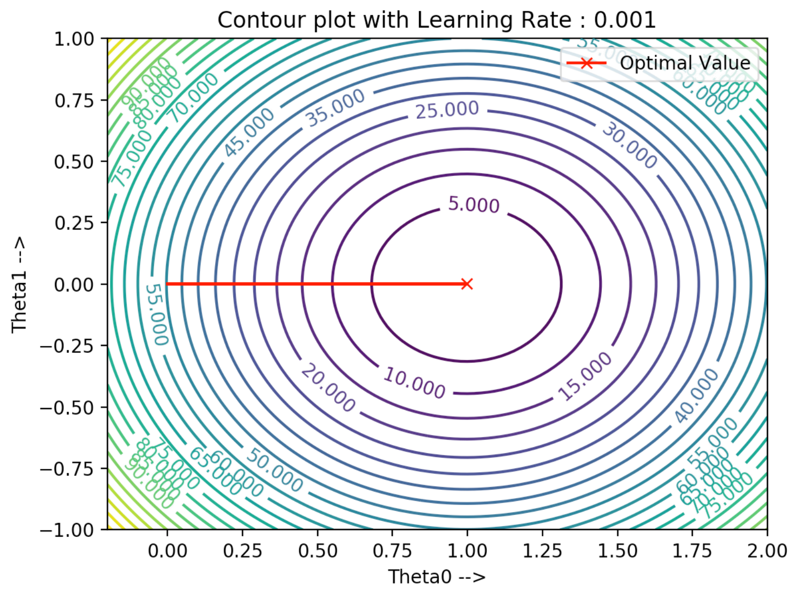
Background has been initialised to the constant blue surface which is the cost function whereas the **foreground** is being continuously updated at each interval to show the new frame. This has been done at a rate of 0.2s for human visualisation.

We can see GD converging to its global minima.



(d) Contours have been plotted with the help of contour function available in the matplotlib library. We need to specify the number of levels and other specifications related to them.

Left side is the diagram of contours of cost function. We can see that as we go away from the centre towards the periphery, contours cluster towards each other showing that the cost function increases much more rapidly as we go away.



Animation of contours has been done in a different way. I have refreshed and updated my graph at each iteration with a gap of 0.2s so as to visualise the moving gradient descent algorithm on the contour.

Red line shows the path followed by the gradient descent.

(e)

Learning Rate(Eta)	Convergence	Oscillatory Steps	Number of Iterations
0.001	Yes	No	122
0.005	Yes	No	22
0.009	Yes	No	8
0.013	Yes	Yes	14
0.017	Yes	Yes	39
0.021	No	Yes	NA
0.025	No	Yes	NA

We can see that as the value of eta increases, our step size in Gradient Descent essentially increases, and we start taking bigger steps. Due to this, it so happens that when we increase the value of eta, we get our convergence at less number of iterations, but when we increase the value of eta above a certain limit, the cost function overshoots and fails to converge in those cases. Here, the error function doesn’t converge for values 0.021 and 0.025, and it essentially overshoots. Also, for eta values of 0.013 and 0.017 we notice that there is an oscillatory behaviour in the start, while for other eta values less than 0.013, it converges smoothly.

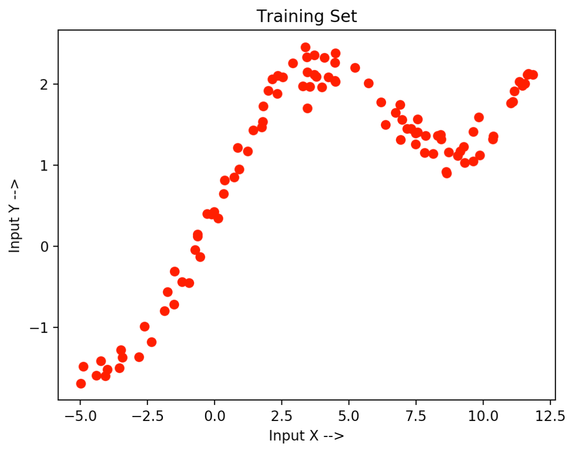
Decreasing Error Value with Iterations

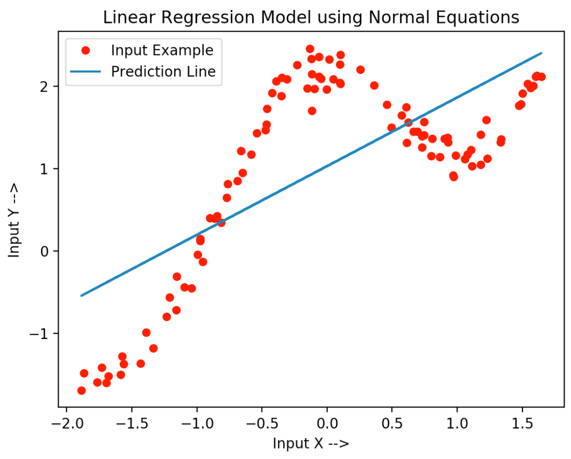
Iteration Count	Error Value
0	49.6627904714
2	37.1503803369
4	27.7904467857
6	20.7887296794
8	15.5510811245
10	11.6330475953
12	8.7021547386
14	6.50969444603
16	4.86962011733
18	3.64275921919
20	2.72500345835
22	2.03847441677
24	1.52491498729
26	1.14074582381
28	0.853367308887
30	0.63839323317
32	0.477581443733
34	0.357285861412
36	0.267298509263
38	0.199983289613
40	0.149628004251
42	0.111959631821
44	0.0837817301119
46	0.0627031941006
48	0.0469353528695
50	0.0351401875432
52	0.0263167904126

54	0.019716430454
56	0.0147790179203
58	0.0110855765502
60	0.00832269030956
62	0.00625590770381
64	0.00470984682134
66	0.00355331287063
68	0.00268816532417
70	0.00204098996309
72	0.00155686913342
74	0.00119472157369
76	0.000923816363727
78	0.000721165176892
80	0.000569571549251
82	0.000456171631391
84	0.000371342594902
86	0.00030788606365
88	0.000260417277899
90	0.000224908157306
92	0.000198345488273
94	0.000178475230312
96	0.000163611243905
98	0.000152492208997
100	0.000144174592585
102	0.000137952582909
104	0.000133298196065
106	0.00012981647263
108	0.000127211962417
110	0.000125263653317
112	0.000123806216779
114	0.000122715978447
116	0.000121900423471
118	0.000121290345933
120	0.000120833976203

122	0.00012049258791
124	0.000120237211711
126	0.000120046177032
128	0.000119903273157
130	0.000119796373626
132	0.000119716407216
134	0.000119656588183
136	0.000119611840435
138	0.000119578366792
140	0.000119553326766
142	0.000119534595525
144	0.000119520583582
146	0.00011951010192
148	0.000119502261091
150	0.000119496395744
152	0.000119492008159
154	0.000119488726017
156	0.000119486270805
158	0.000119484434178
160	0.000119483060285
162	0.000119482032542
164	0.000119481263737
166	0.000119480688631
168	0.000119480258422
170	0.000119479936603

2. Data has been normalised to a zero mean and unit variance as a preprocessing step in this part. Similar to first part, we obtain our design matrix X (after appending ones) and n+1 dimensional vector y. We created a method which given the values of X and y, will give us the optimal values of theta (Normal Equation Implementation).





(a) This is the sample training data, and the straight line fit as per the normal equations described in the class. The straight line highly underfits the training data and has a very poor accuracy when it comes to generalisation. Hence, this unweighted linear regression fails.

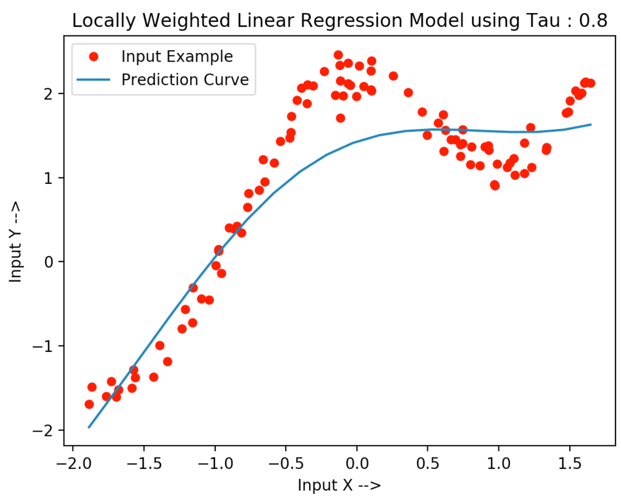
(b)
To get the analytical solution of θ in this new setting, we proceed as follows:

$$\theta = (X^T W X)^{-1} X^T W Y$$

Here, W is the diagonal square matrix of size (n + 1) by (n + 1). W_{ii} refers to the ith entry in diagonal and the value is given by:-

$$W_{ii} \;=\; e^{-\frac{(x - x_i)^T (x - x_i)}{2\tau^2}}$$

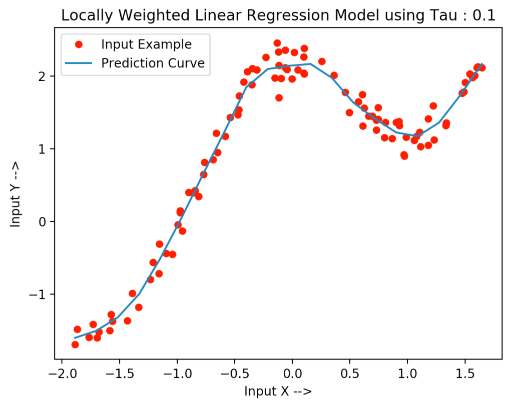
We sampled out 20 points on the x-axis and used them as the value of x so as to get the approximate curve. For each such x, we computed the value of optimal parameters using the closed form equation as we derived above, and then used this x to get the prediction at that value of x with those parameters. Hence, each such x will give me those many value of predictions which can be used to draw small line segments and the approximate curve. The parameter tau decided the bandwidth of this curve.

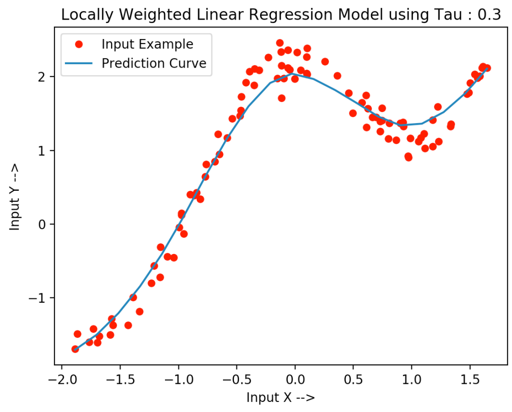


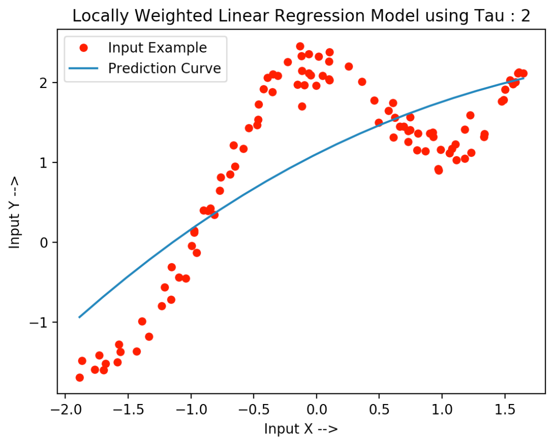
In comparison to the linear model, this is a much better model as it is able to fit the curve better than the former model which was a disaster.

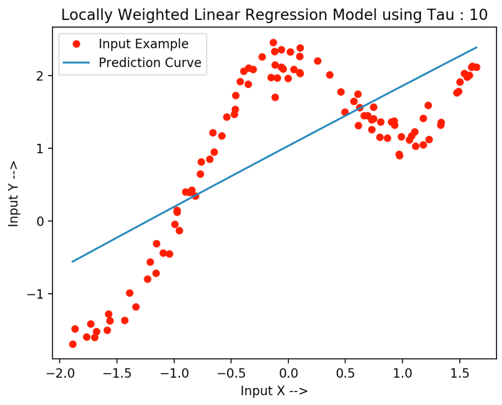
Blue line show the prediction line we got as a result of LWR Model.

(c)









τ	Fitting	Training Set accuracy	Test data accuracy
0.1	Overfits	Good	Poor
0.3	Good Fitting	Good	Good
0.8	Underfits	Poor	Poor
2	Underfits	Poor	Poor
10	Underfits	Poor	Poor

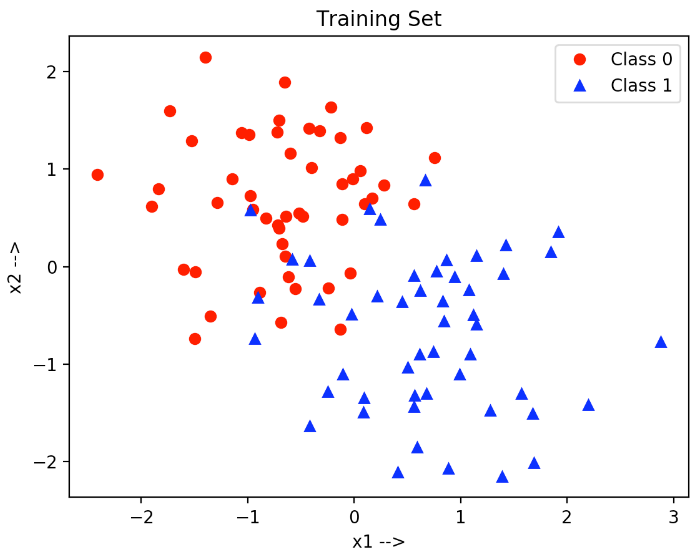
At a very low value of τ = 0.1, we can see that the prediction model overfits over the data points, and hence it will not be a good choice since it will fail to generalise to new examples. At a high value of τ like 2 or 10, the model underfits and it will fail on the training set itself and as well as on new examples.

Better value of τ that can be used as a bandwidth parameter is 0.3, and it will generalise to new examples as well.

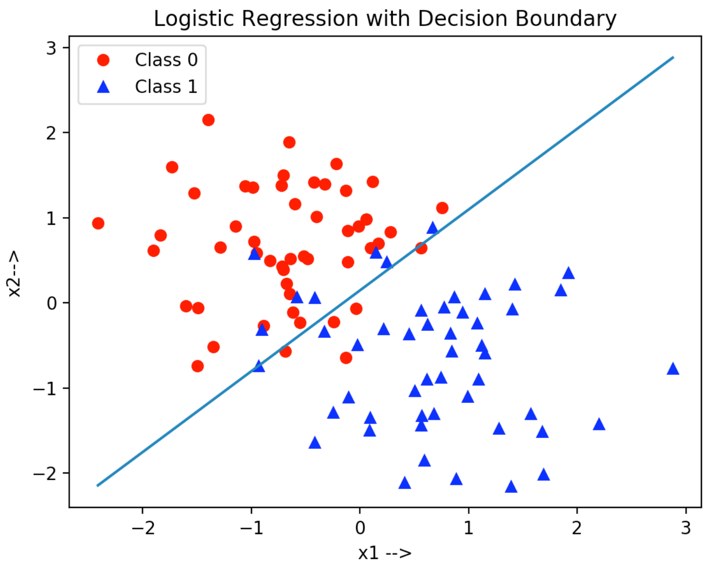
Best choice of τ = 0.3

3.
We defined function to compute the sigmoid, hypothesis, hessian, gradient, and likelihood. Before that, we load the matrix X and vector y after mean normalisation and feature scaling. Don't forget to add the column of ones to X before you run the Newton-Raphson Method.

(a)
Number of iterations taken to converge = 8
Converging Criteria = $\max_j|(\theta(t+1) - \theta(t))| < \epsilon$ => Algorithm will converge when the value of maximum component of the absolute difference becomes less than some certain epsilon.
Here, $\epsilon = 10^{-10}$
Optimum Values of θ resulting from this method = $[0.401, 2.589, -2.7256]^T$
Hypothesis function = $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$



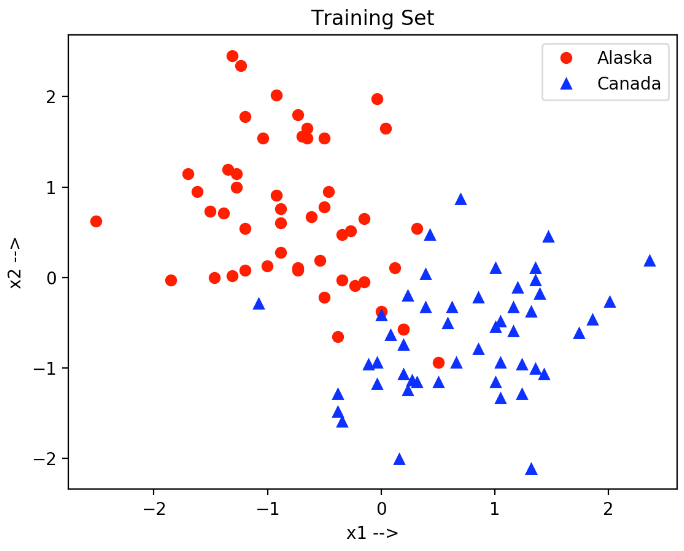
(b) Figure given below shows the decision boundary which in our case is a straight line with the parameters computed using this method. This method converges in 8 iterations which makes this method much faster than Gradient Descent.



4.
Here we simply load the matrix X, and normalise the data values in each column. The vector y is an array of strings, so we map the string to binary values 0 and 1, where *Class 0 refers to 'Alaska' and Class 1 refers to 'Canada'.*
(a) Using the closed form equations as described in the class, GDA gives us the following values of the parameters.

$$\phi = 0.5$$
$$\mu_0 = [-0.7553\ ,\ 0.6851]$$
$$\mu_1 = [0.7553\ ,\ -0.6851]$$
$$\Sigma = \begin{bmatrix} 0.42953048 & -0.02247228 \\ -0.02247228 & 0.53064579 \end{bmatrix}$$

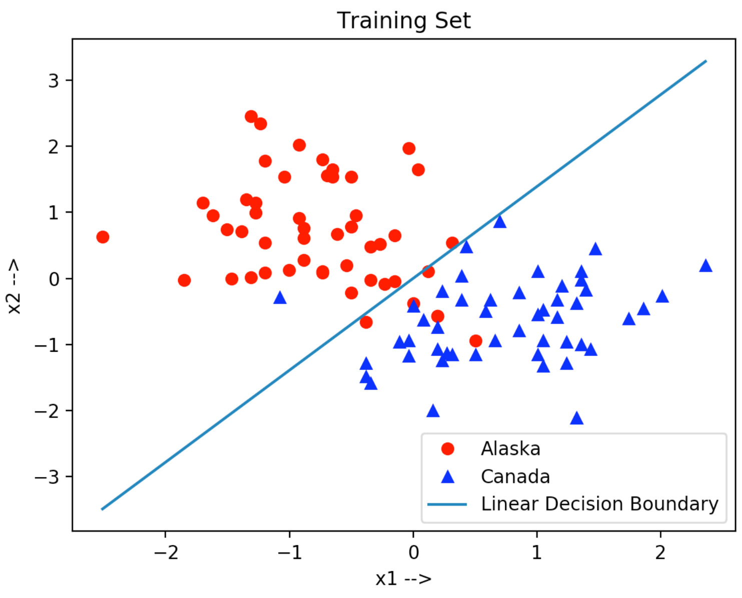
(b) Figure depicting the two classes - Alaska(0) and Canada(1) after normalisation.



(c) Equation of decision boundary in terms of μ_0, μ_1 *and* Σ is:

$$x = [x_1\ x_2]^T$$

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



The equation of DB in this case is :
-3.39x + 2.44y = 1.11022302463e-15 ≅ 0

Here, ϕ is the the parameter of Bernoulli distribution, and tells us the probability of training example to occur with Class-1 (i.e. Canada)

Since both the covariances are the same, we will get a straight line with the equation as given above.

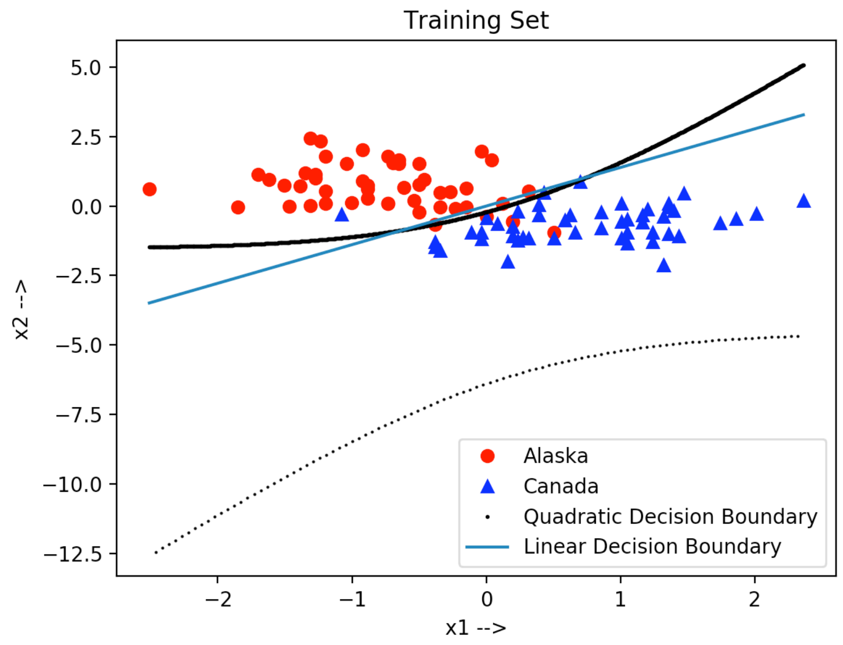
(d)

$$\phi = 0.5$$
$$\mu_0 = [-0.7553\ ,\ 0.6851]$$
$$\mu_1 = [0.7553\ ,\ -0.6851]$$
$$\Sigma_0 = \begin{bmatrix} 0.38158978 & -0.15486516 \\ -0.15486516 & 0.64773717 \end{bmatrix}$$
$$\Sigma_1 = \begin{bmatrix} 0.47747117 & 0.1099206 \\ 0.1099206 & 0.41355441 \end{bmatrix}$$

(e) The equation for the quadratic boundary separating the two regions is:

$x = [x_1\ x_2]^T$

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



Plotting the boundary means solving the equation for y in terms of x. Let x = [x y]^T
Setting the probability of y = 1 given x to the value of y = 0 given x, we have

P(y = 1/x) = P(y = 0/x)

Plugging in the values of these probabilities by Bayes Theorem and simplifying it further we get,

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

The right side of the equation is a constant term which we can call C, this will give

$$C = \log|\mathbf{\Sigma_0}| - \log|\mathbf{\Sigma_1}| + 2\log\phi - 2\log(1 - \phi)$$

And the equation to solve becomes

$$(\mathbf{x} - \mu_1)^{\mathbf{T}}\mathbf{\Sigma_1^{-1}}(\mathbf{x} - \mu_1) - (\mathbf{x} - \mu_0)^{\mathbf{T}}\mathbf{\Sigma_0^{-1}}(\mathbf{x} - \mu_0) = C$$

Put x as [x y], and the values of Σ_1^{-1} and Σ_0^{-1} as shown below

$$[x_1 \quad y_1]\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} - [x_0 \quad y_0]\begin{bmatrix} p & q \\ r & s \end{bmatrix}\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = C$$

Where $x_0 = x - \mu_{00}, y_0 = x - \mu_{01}, x_1 = x - \mu_{10}, y_1 = x - \mu_{11}$, and simplifying it further we get an equation of the form:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$ Where A,B,C,D,E,F are constants

Value of these constants come out to be something like this:
A = -0.671347799363
B = -2.57367267331
C = 0.865931980622
D = -7.61570638708
E = 5.71934612537
F = 0.489254753463

(f)
Linear Decision Boundary almost passes through the origin in the normalised data.
When we compute the Discriminant($D = B^2 - 4AC$), we get D ≅ 8.9492 > 0, and we can then conclude that the resulting conic section has to be a hyperbola. On plotting the quadratic boundary, this indeed comes out to be a hyperbola as we can see in the earlier figure. The data points between the arcs of the hyperbola belongs to the Class-1(Canada), and the data points outside the two branches belong to the class-0(Alaska).
It can be seen that both the models have a good degree of accuracy when it comes to classification. However, the quadratic boundary is slightly more accurate by 2 or 3 examples, and implies that the underlying distribution modelling the data is gaussian. As we conclude that both the classes can't have the same covariance matrix possibly because that Σ_0 has a negative covariance while Σ_1 has a positive covariance, and modelling both of them with the same covariance(Σ) isn't very accurate.