COL774: Machine Learning. Assignment 1

Arpan Mangal — 2016CS10321

February 2019

1 Linear Regression

1.1 Batch Gradient Descent

Batch gradient descent for optimizing $J(\theta)$ was done using the vectorised implementation of gradient descent. First data was read and converted to matrices (adding the intercept term to X), giving $X \in \mathbb{R}^{m\times 2}$ and $Y \in \mathbb{R}^m$, where $m = size(data) = 100$. $\theta \in \mathbb{R}^2$.

Data was not divided into test and validation set, since the dataset was too small and partition would have caused loss of data. Also having only one feature rules out possibility of high overfitting.

$$J(\theta) = \frac{1}{2m}(Y - X\theta)^T(Y - X\theta) \tag{1}$$

$$\frac{\partial J}{\partial \theta} = \frac{1}{m}X^T(X\theta - Y) \tag{2}$$

$$\theta := \theta - \eta\frac{\partial J}{\partial \theta} \tag{3}$$

1. Learning Rate $(\eta) = 1$
2. Convergence Criteria: Change in cost is less than 0.01%.
3. Parameters: $\theta = \begin{pmatrix} 0.98997 \\ 0.00082 \end{pmatrix}$

The data was normalised for training. It was reset while plotting. The reported θ is for unnormalised data.

1.2 Data Plots

Following hypothesis function was learnt. (Figure 1)

The optimal fit (based on analytical solution) slightly differs from this plot, as training was stopped when change in cost was less then 0.01% (convergence criterion), to avoid overfitting.

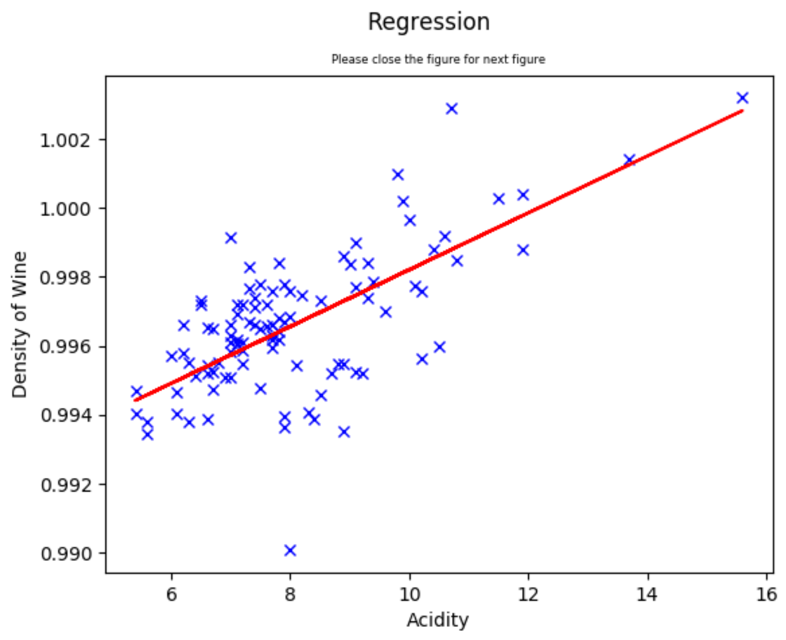


Figure 1: $y = \theta^T x$

1.3 3D Mesh

Real time 3D plot of $J(\theta)$ with gradient descent: (Figure 2)

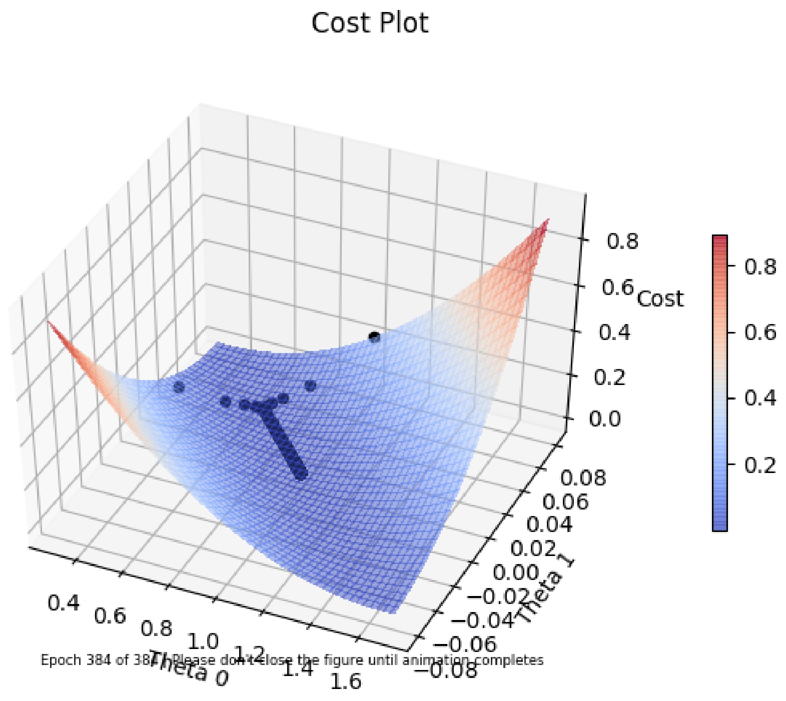


Figure 2: $J(\theta)$

1.4 Contour Plot

Real time contour plot of $J(\theta)$ with gradient descent: (Figure 3)

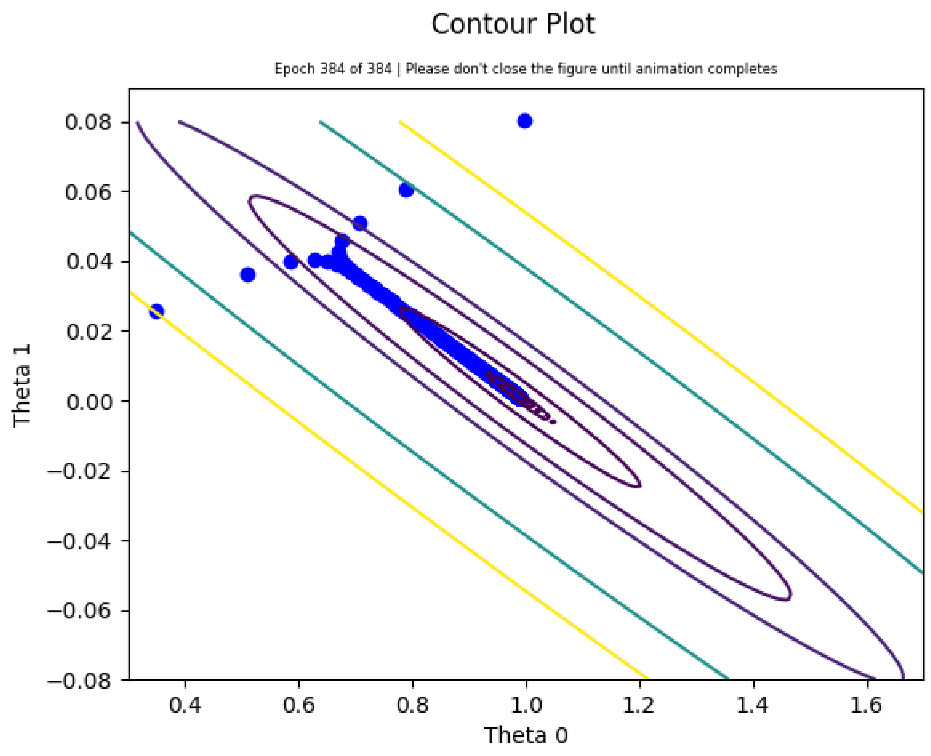


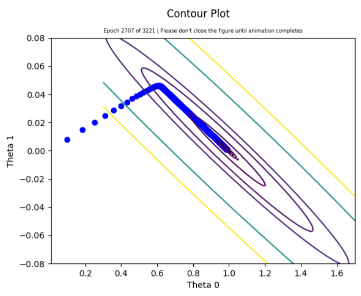
Figure 3: $J(\theta)$

1.5 Contours

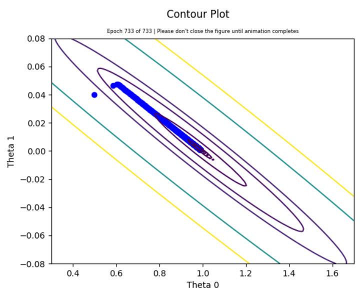
Contour plots with different learning rates, $\eta = \{0.1, 0.5, 0.9, 1.3, 1.7, 2.1, 2.5\}$. (Figure 4)

At small η like $\eta = 0.1$, the algorithm proceeds slowly and approaches the minimum taking really small steps. In intermediate range like $\eta = 0.9$, it begins oscillating (initially), but eventually falls along the gradient and converges.

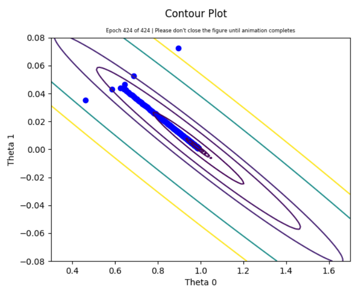
After $\eta = 1$, the gradient descent algorithm starts diverging. At $\eta = 1.3$, initially it continues converging, but the oscillation is high and it eventually overshoots the minima and starts diverging. Though divergence is expected at high learning rate, in this case part of the reason is approximate normalisation in the implementation (done to easily de-normalise X and θ), due to which it diverges beyond learning rates like $\eta = 1.3$.



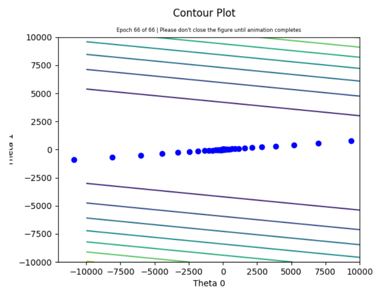
(a) $\eta = 0.1$



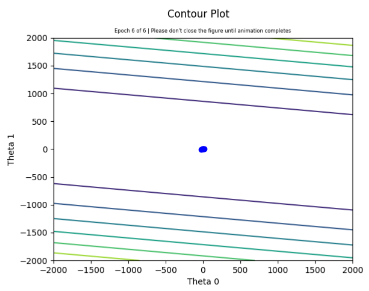
(b) $\eta = 0.5$



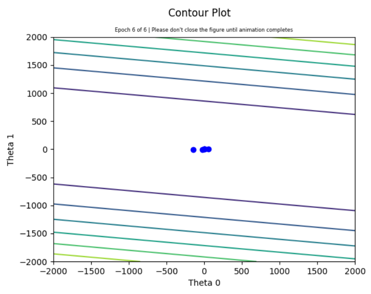
(c) $\eta = 0.9$



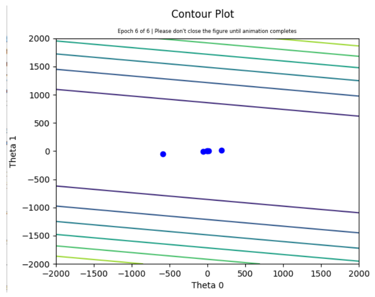
(d) $\eta = 1.3$



(e) $\eta = 1.7$



(f) $\eta = 2.1$



(g) $\eta = 2.5$

Figure 4: Contours

2 Locally Weighted Linear Regression

2.1 Unweighted Linear Regression

Analytical solution was used to find the optimal value of θ. First data was read and converted to matrices (adding the intercept term to X), giving $X \in \mathbb{R}^{m\times 2}$ and $Y \in \mathbb{R}^{m}$, where $m = size(data) = 100$. $\theta \in \mathbb{R}^2$.

Again, data was not divided into test and validation set, since the dataset was too small and partition would have caused loss of data.

Data was also not normalised since it's not required in the case of Analytical solution.

$$J(\theta) = \frac{1}{2m}(Y - X\theta)^T(Y - X\theta) \tag{4}$$

$$\theta = (X^T X\theta)^{-1}X^T Y \tag{5}$$

1. Parameters: $\theta = \begin{pmatrix} 0.328 \\ 0.175 \end{pmatrix}$

Corresponding plot is shown in Figure 5.

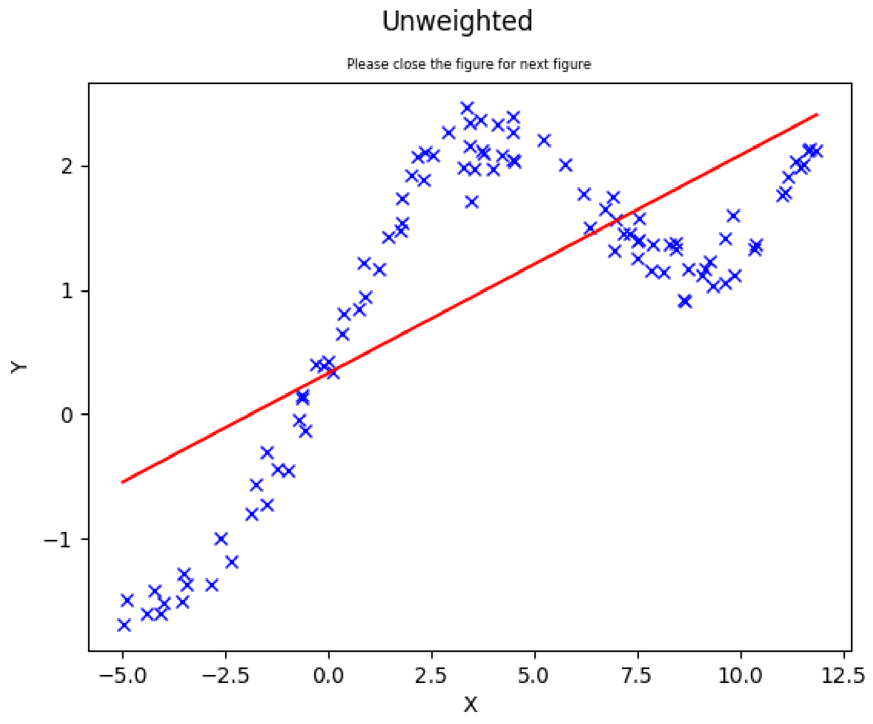


Figure 5: Unweighted Linear Regression

2.2 Locally Weighted Linear Regression

Analytical solution was used to find the optimal value of θ.

For each data point, the diagonal weight matrix W is calculated w.r.t. that point. Then prediction is made using the data points, and plotted. Finally all the points are joined by a smooth line. (Since the dataset is small 100 points, the same set was used as validation set as well as training set)

$$J(\theta) = \frac{1}{2m}(Y - X\theta)^T W (Y - X\theta) \tag{6}$$

$$\theta = (X^T W X\theta)^{-1} X^T W Y \tag{7}$$

$$W_{ii} = exp(-\frac{(x - x^{(i)})^2}{2\tau^2}), W_{ij} = 0 \forall i \neq j, W \in \mathbb{R}^{m \times m} \tag{8}$$

For $\tau = 0.8$ we get a nicely fitting curve (Figure 6).

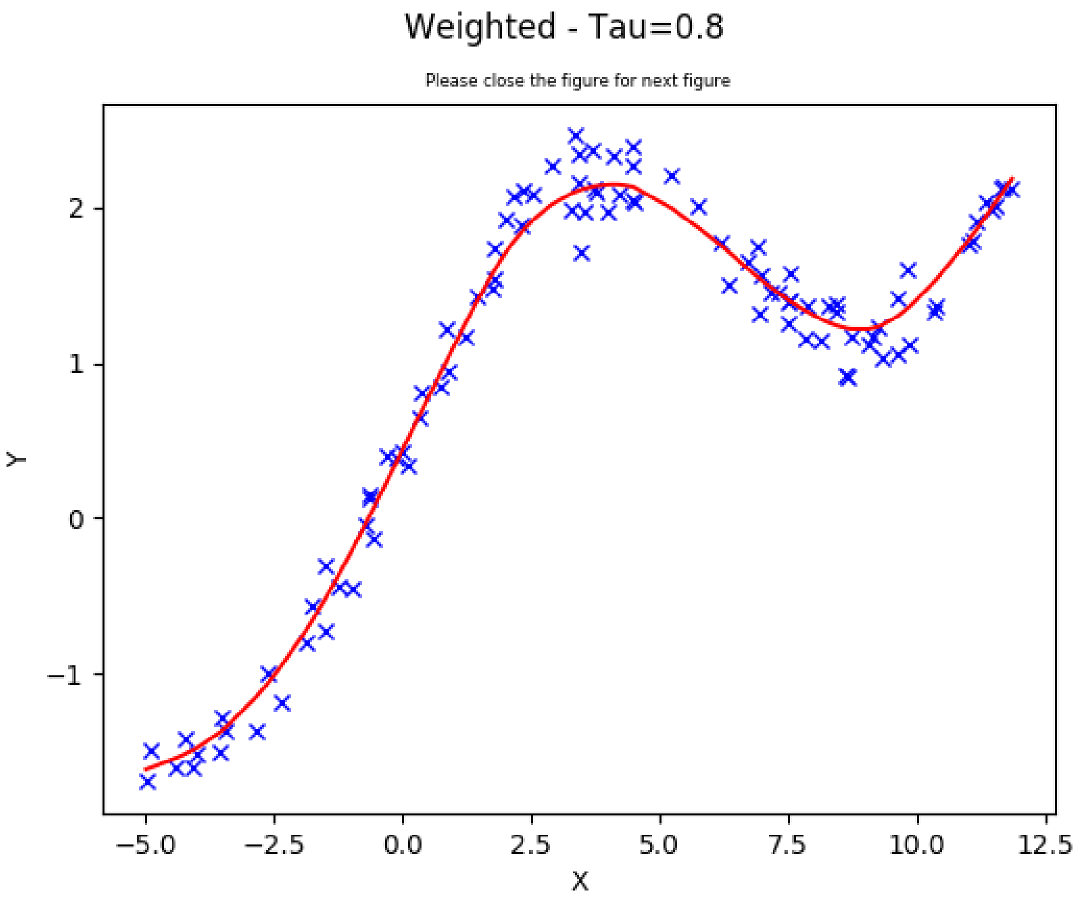


Figure 6: Weighted Linear Regression – $\tau = 0.8$

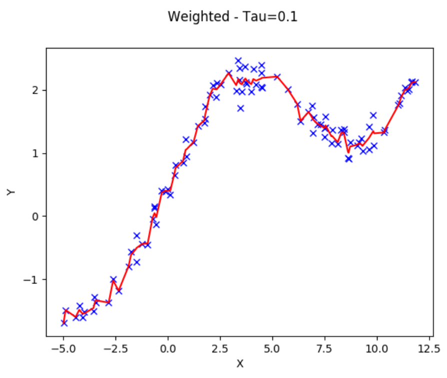
2.3 Weighted – Varying τ

Plots for $\tau = 0.1, \tau = 0.3, \tau = 2, \tau = 10$ (Figure 7)

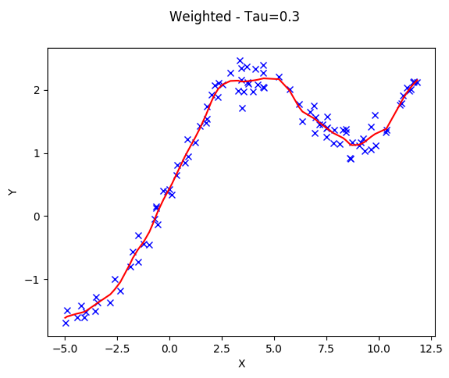
1. Among the 5 values, $\tau = 0.8$ gives the best fit through the data. After $\tau = 0.8$, $\tau = 2.0$ is the best (considering overfitting to be bad).

2. As $\tau \to 0$, $W_{ii} \to 0$, i.e. all the points $x_{(i)}$, away from point x will be given weights close to 0, leading it to overfitting the training data. This is observed in the $\tau = 0.8$ curve.

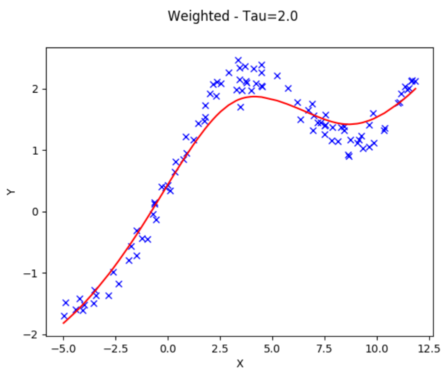
3. As $\tau \to \infty$, $W_{ii} \to 1$, which makes it exactly same as unweighted linear regression. This is observed in the $\tau = 10$ curve, in which the function is close to a straight line.



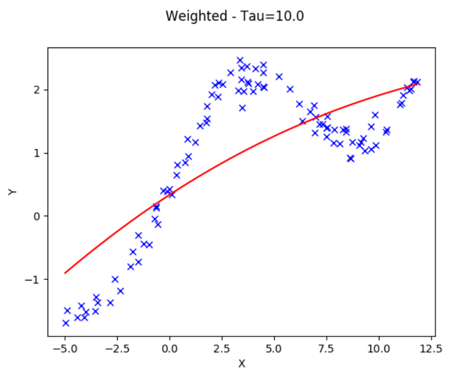
(a) $\tau = 0.1$



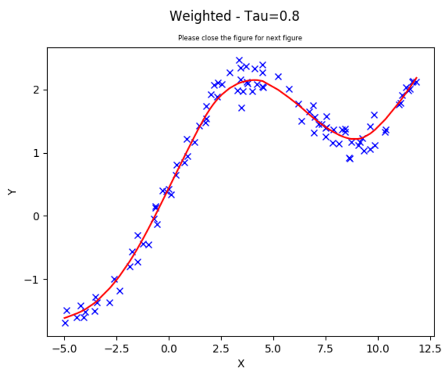
(b) $\tau = 0.3$



(c) $\tau = 2.0$



(d) $\tau = 10.0$



(e) $\tau = 0.8$

Figure 7: Weighted Linear Regression

3 Logistic Regression

3.1 Newton's Method

Newton's method was used for optimising $L(\theta)$. First data was read and converted to matrices (adding the intercept term to X), giving $X \in \mathbb{R}^{m\times 3}$ and $Y \in \mathbb{R}^m$, where $m = size(data) = 100$. $\theta \in \mathbb{R}^3$.

Data was not normalised as not required when directly calculating Gradient and Hessian.

Gradient and Hessian were calculated via vectorised implementations.

$$\nabla_\theta L(\theta) = X^T(Y - \sigma(X\theta)) \tag{9}$$

$$H = X^T D X \tag{10}$$

where,

$$D_{ii} = \sigma(X\theta)(1 - \sigma(X\theta)), D_{ij} = 0 \; \forall \; i \neq j, D \in \mathbb{R}^{m \times m} \tag{11}$$

$$\theta := \theta - H^{-1}\nabla_\theta L(\theta) \tag{12}$$

The process gives right set of parameters in just 1-2 iterations. More iterations make H singular.

Parameters: $\theta = \begin{pmatrix} 0.142 \\ -0.680 \\ 0.663 \end{pmatrix}$

3.2 Plot

Plotting $h_\theta(x) = 0.5$ gives us a straight line logistic plot. (Figure 8)

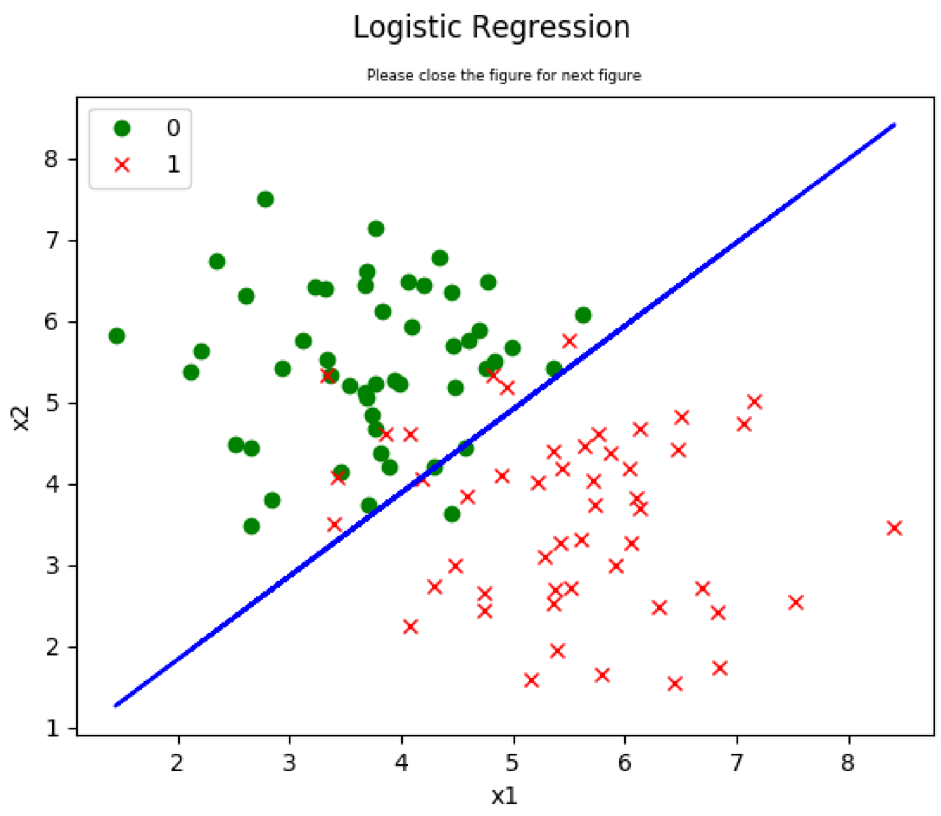


Figure 8: Logistic Regression

4 Gaussian Discriminant Analysis

4.1 GDA – common co-variance matrix

First data was read and converted to matrices, giving $X \in \mathbb{R}^{m\times 2}$ and $Y \in \mathbb{R}^{m}$, where $m = size(data) = 100$.

The model parameters (ϕ, μ_0, μ_0, Σ) were computed.

Parameter values:

1. $\phi = 0.5$
2. $\mu_0 = \begin{pmatrix} 98.38 \\ 429.66 \end{pmatrix}$
3. $\mu_1 = \begin{pmatrix} 137.46 \\ 366.62 \end{pmatrix}$
4. $\Sigma = \begin{pmatrix} 287.482 & -26.748 \\ -26.748 & 1123.25 \end{pmatrix}$

Data was not normalised as not required. Variance will be automatically captured in the co-variance matrix.

4.2 Training Data Plot

Plot of training data (Figure 9)

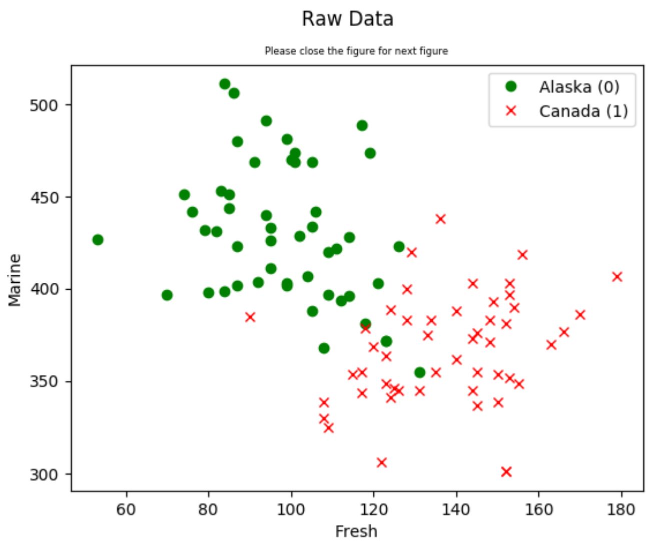


Figure 9: Training Data

4.3 GDA boundary

In this case the both $x1$ and $x2$ have common co-variance matrix, and taking $\phi = 0.5$ (which is true in our case),

the equation of the boundary is given by:

$$(x - \mu_0)^T \Sigma^{-1} (x - \mu_0) - (x - \mu_1)^T \Sigma^{-1} (x - \mu_1) = 0 \tag{13}$$

which is a linear equation, since both expressions have the same Σ. Note that, $x = \begin{pmatrix} x1 \\ x2 \end{pmatrix}$

This decision boundary is plotted in Figure 10.

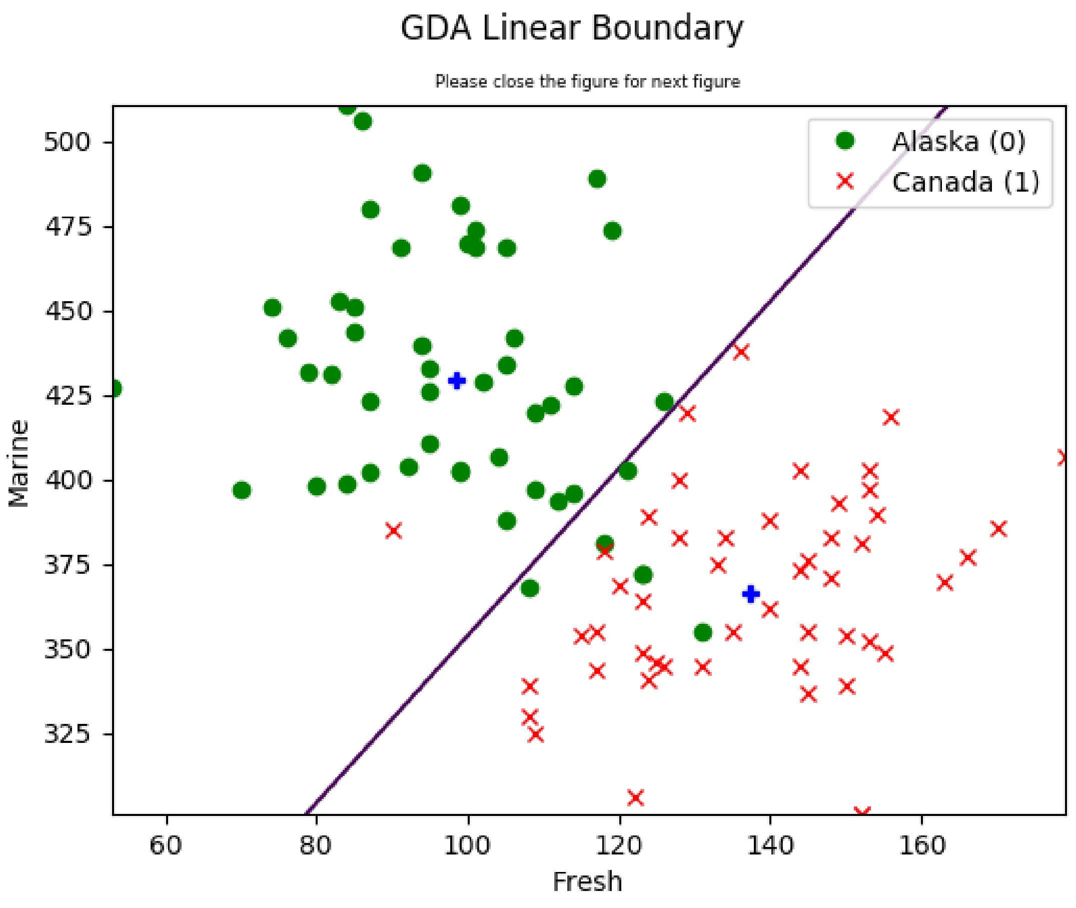


Figure 10: GDA Linear Boundary

4.4 GDA – different co-variance matrices

The model parameters (ϕ, μ_0, μ_0, Σ_0, Σ_1) were computed, using respective expressions.

Parameter values:

1. $\phi = 0.5$

2. $\mu_0 = \begin{pmatrix} 98.38 \\ 429.66 \end{pmatrix}$

3. $\mu_1 = \begin{pmatrix} 137.46 \\ 366.62 \end{pmatrix}$

4. $\Sigma_0 = \begin{pmatrix} 255.396 & -184.331 \\ -184.331 & 1371.104 \end{pmatrix}$

5. $\Sigma_1 = \begin{pmatrix} 319.568 & 130.835 \\ 130.835 & 875.396 \end{pmatrix}$

Data was not normalised as not required. Variance will be automatically captured in the co-variance matrix.

4.5 GDA boundary

The equation of the boundary is given by:

$$p(y^{(i)} = 1|x^{(i)}) = 0.5 \tag{14}$$

or,

$$p(y^{(i)} = 1|x^{(i)}) = p(y^{(i)} = 0|x^{(i)}) \tag{15}$$

which when expanded,

$$\frac{(1-\phi)}{\det(\Sigma_0)} exp(-\frac{(x-\mu_0)^T {\Sigma_0}^{-1}(x-\mu_0)}{2}) - \frac{\phi}{\det(\Sigma_1)} exp(-\frac{(x-\mu_1)^T {\Sigma_1}^{-1}(x-\mu_1)}{2}) = 0 \tag{16}$$

Taking the constants into the exponents, and then equating the power of the exponents, we get a quadratic equation. This equation was directly plotted in python by plotting the 0^{th} contour of the expression.

This decision boundary is plotted in Figure 11.

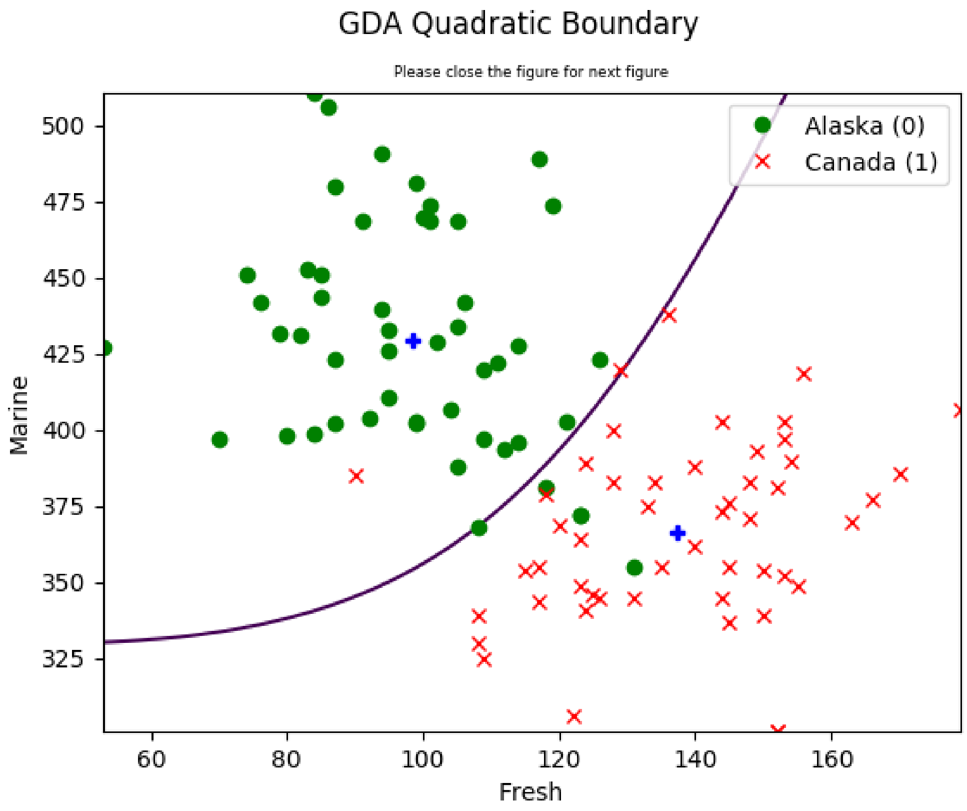


Figure 11: GDA Quadratic Boundary

4.6 Boundaries

Both linear as well as the quadratic boundaries seem to separate the data well, giving almost the same result on our dataset. Since the dataset is small it is difficult to comment which is better, though quadratic is likely to perform better as it has different co-variances for different classes.

However zooming out the quadratic boundary (Figure 12), shows us a hyperbolic boundary. We can see that the data points just lie close to one of the bounding curves, but still the quadratic boundary would predict points below the second curve as ”Alaska” while they should be ”Canada” more probably as per our small training set.

Thus in this case the linear boundary serves the purpose of prediction better. This validates the general belief that GDA will perform better as we have more and more data, while logistic regression (or GDA linear boundary) would perform better in case of limited data.

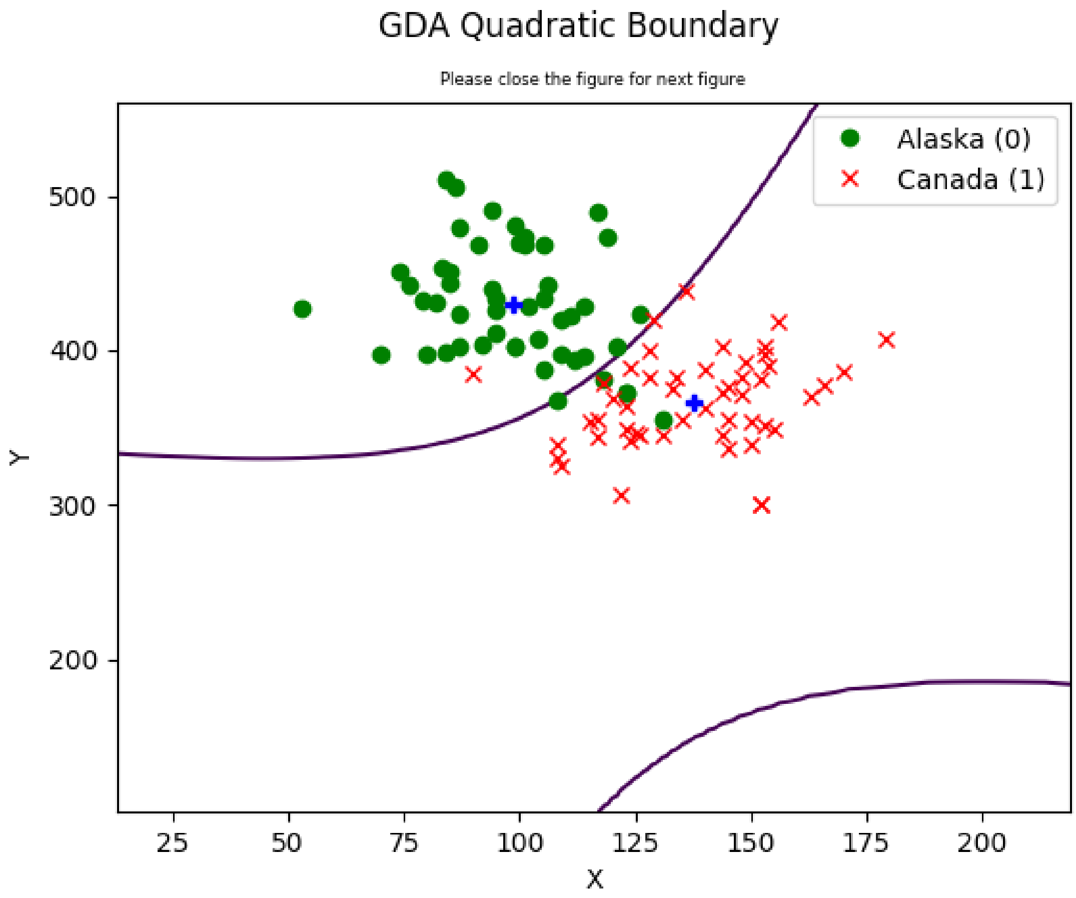


Figure 12: Quadratic Boundary – Zoomed out