Machine Learning
COL774 - Lab Assignment 1
Supervisor - Prof. Parag Singla

Shreshth Tuli
2016CS10680
Date: 12/02/2019

Introduction

In this assignment, we implemented Simple Supervised Machine Learning algorithms on Python. The algorithms and models were tested using real-life experimental data for accuracy and efficacy. We present different models, design decisions, parameters, hyper parameter tuning strategies used.

Linear Regression

Linear Regression is a supervised learning algorithm that allows us to fit a linear curve (i.e. a straight line) on given set of training examples.

- Hypothesis function : Linear

$$h_\theta(x) = \theta^T \cdot x \tag{1}$$

- Optimisation Strategies used : Gradient Descent
- Variants of Gradient Descent : Batch Gradient Descent, Stochastic Gradient Descent
- Error function : Mean Square error

$$\begin{aligned} J_{BGD}(\theta) &= \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 & (2)\\ &= \frac{1}{2m}(Y - X\theta)^T(Y - X\theta) & (3)\\ J_{SGD}(\theta) &= \frac{1}{2}(y^{(i)} - h_\theta(x^{(i)}))^2 & (4)\\ &= \frac{1}{2m}(Y - X\theta)^T(Y - X\theta) & (5) \end{aligned}$$

- Batch gradient descent:

$$\begin{aligned} \theta^{t+1} &= \theta^t - \nabla J_{BGD}(\theta^t) & (6)\\ &= \theta^t - \frac{1}{m}X^T(X\theta - Y) & (7) \end{aligned}$$

- Stochastic gradient descent:

$$\begin{aligned} \theta_j^{t+1} &= \theta_j^t - \nabla J_{SGD}(\theta_j^t) & (8)\\ &= \theta_j^t - (y^{(i)} - h_{\theta_j}(x^{(i)}))x_j^{(i)} \quad \forall j & (9) \end{aligned}$$

- Threshold : 1×10^{-20}
- Termination condition : Change in error function $<$ threshold for atleast 3 consecutive iterations

Solutions:

- Analytical solution: [[0.9966201] [0.0013402]] Gradient Descent solution: [[0.9966201] [0.0013402]]

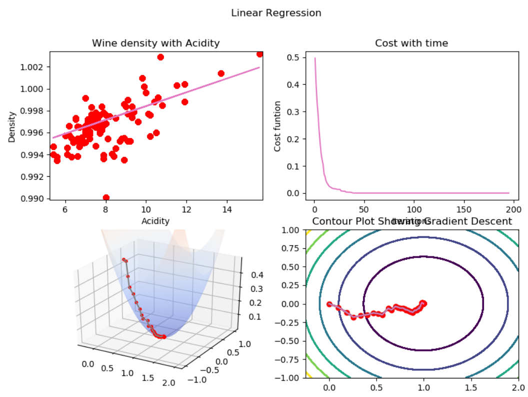
Figures 1 - 2 show the fitted line, cost with number of iterations, cost as 3D convex function. and contours for different approaches. For learning rate of > 2.1, the gradient descent algorithm diverges. The analytical solution was obtaines as:

$$\theta_{analytical} = (X^TX)^{-1}X^TY \tag{10}$$

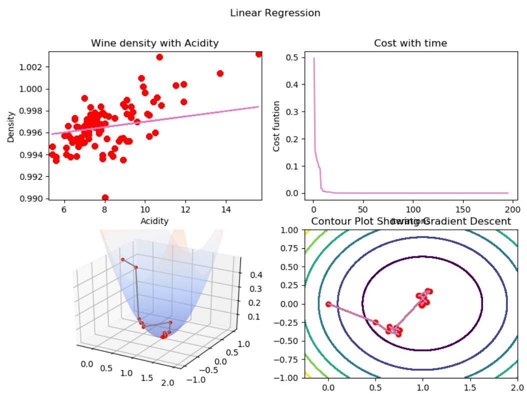
The number of iterations and time taken for different descent variations are shown in Table 1.

Gradient Descent	Learning Rate	Time (seconds)	Number of iterations
BGD	0.1	0.006999	216
	0.5	0.002000	37
	0.9	0.000001	14
	1.3	0.000999	23
	1.7	0.001999	71
	2.1	not converged	not converged
	2.5	not converged	not converged
SGD	0.1	0.005897	268
	0.5	0.001099	297
	0.9	0.000090	198
	1.3	not converged	not converged
	1.7	not converged	not converged
	2.1	not converged	not converged
	2.5	not converged	not converged

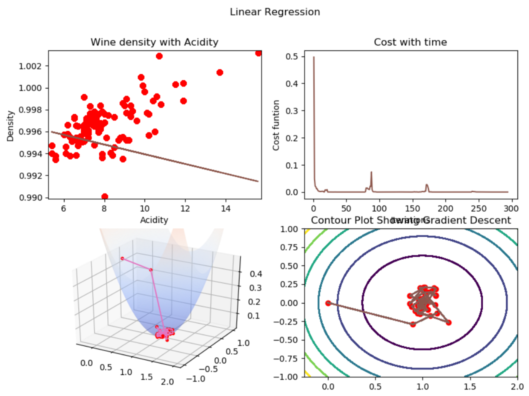
Table 1: Time and number of iterations for different descent algorithms



(a) Learning rate = 0.1

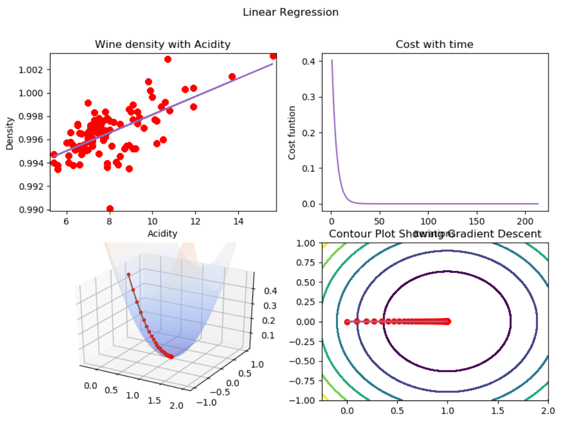


(b) Learning rate = 0.5

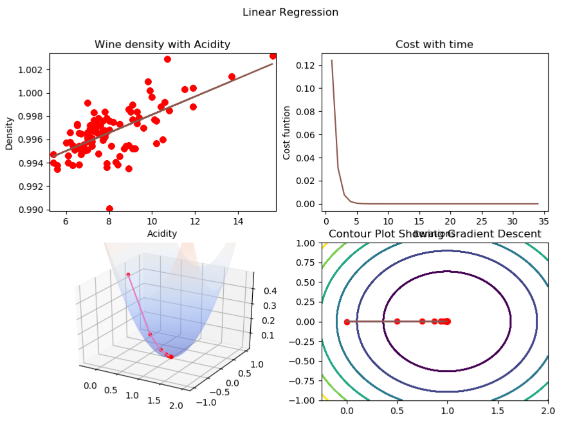


(c) Learning rate = 0.9

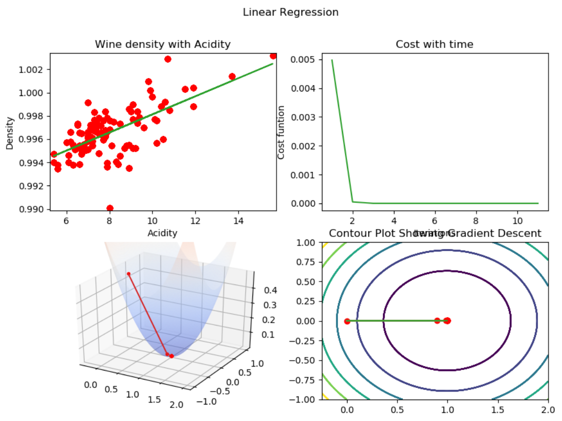
Figure 1: Plots for different learning rate - Stochastic Gradient Descent



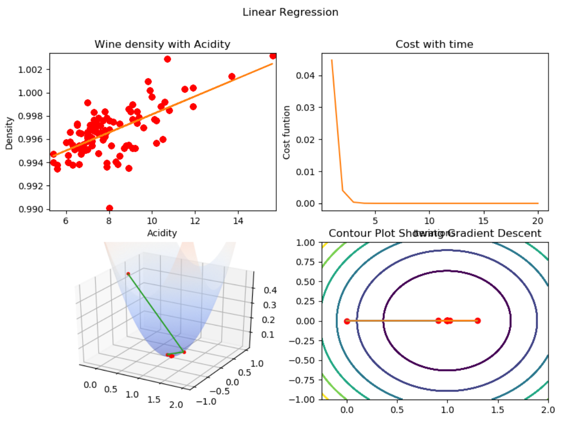
(a) Learning rate = 0.1



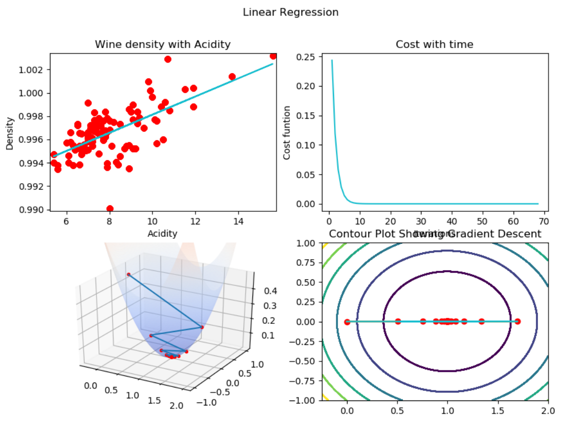
(b) Learning rate = 0.5



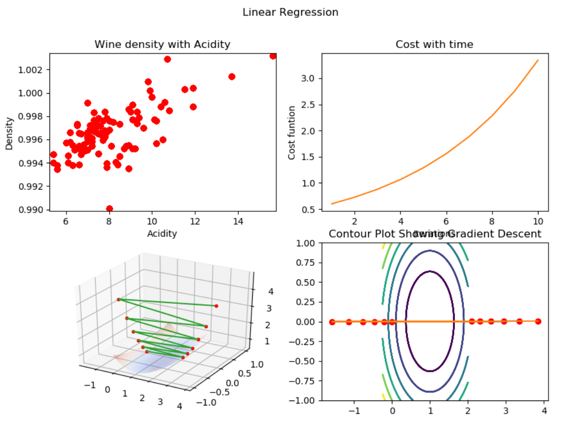
(c) Learning rate = 0.9



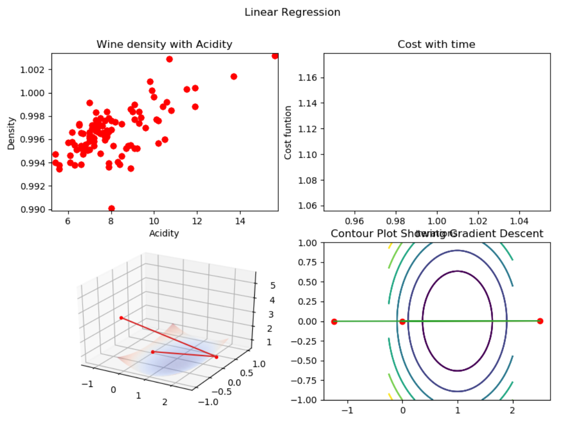
(d) Learning rate = 1.3



(e) Learning rate = 1.7



(f) Learning rate = 2.1



(g) Learning rate = 2.5

Figure 2: Plots for different learning rate - Batch Gradient Descent

Observations:

1. As we first increase the learning rate the number of iterations decreases significantly from 216 to 14, as we increase learning rate from 0.1 to 0.9. This is becuase we are taking much larger steps as we increase the learning rate.
2. As we increase learning rate further, the number of iterations increases and the parameter θ jumps to other side of the minima.
3. After 1.7, the gradient descent algorithm diverges as the steps are so large that it starts moving away from the minima.
4. The best learning rate for batch gradient descent is 0.9 as it converges in least iterations.

Locally Weighted Linear Regression

Locally weighted regression allows us to fir a non linear curve to a supervised training set. We can learn a linear approximate function for each point x by optimizing the following cost function:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} w^{(i)}(y^{(i)} - h_\theta(x^{(i)}))^2 \tag{11}$$

$$= \frac{1}{2m}(Y - X\theta)^T W(Y - X\theta) \tag{12}$$

$$where\ w^{(i)} = e^{\frac{-(x^{(i)}-x)^2}{2\tau^2}} \tag{13}$$

$$and\ W_{ij} = w^{(i)}\ for\ i = j,\ 0\ otherwise \tag{14}$$

- Optimization techniques used : Batch gradient descent, Stochastic gradient descent and Normal Equations (Un-normalized data)
- Descent equation:

$$\theta^{t+1} = \theta^t - \nabla J(\theta^t) \tag{15}$$

$$= \theta^t - \frac{1}{m}X^T W(X\theta - Y) \tag{16}$$

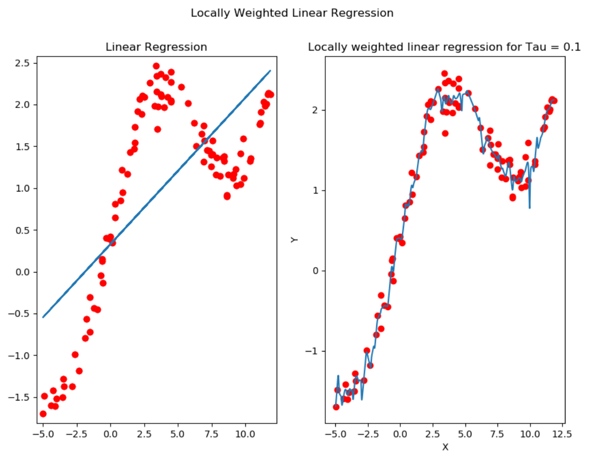
- Analystical Soltion equation:

$$\theta_{analytical} = (X^T W X)^{-1} X^T W Y \tag{17}$$

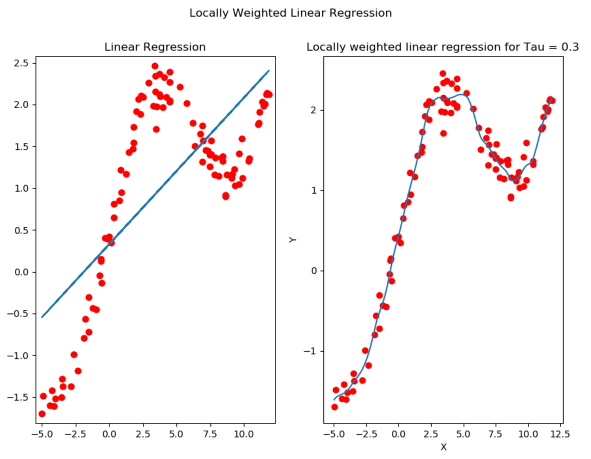
- Parameter : τ called the bandwidth parameter

Observations:

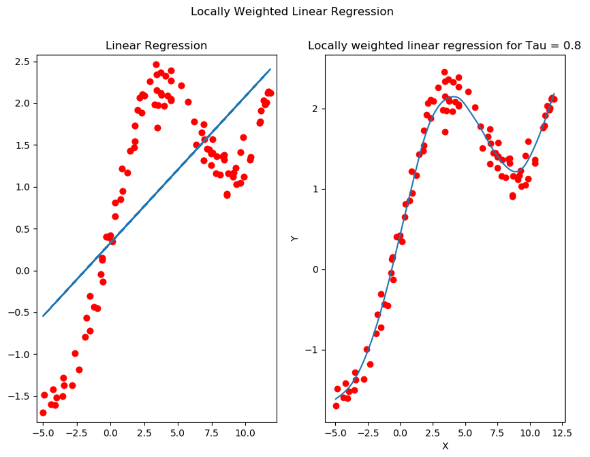
- We observe that for low bandwidth parameter, the fit for each training example is much closer than when bandwidth parameter is higher.
- When bandwidth parameter is less than 0.3 the the curve overfits the data. This is beacuse, when τ is low, the weights $w^{(i)}$ is given most to the nearest example to x and falls of with distance very rapidly.
- When bandwidth parameter is higher than 0.8 then the curve underfits the data. As τ becomes higher and higher, the curve becomes more and more like straight line and for $\tau = 100$ the curve is almost a straight line. This is because the weights $w^{(i)}$ are all constant as τ increases.
- The optimal bandwidth parameter is 0.8



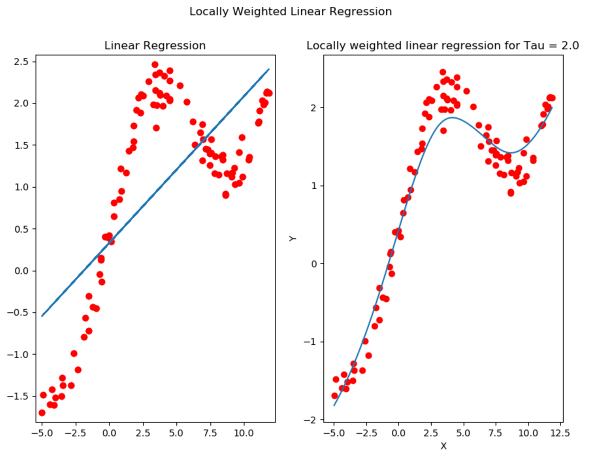
(a) Bandwidth parameter = 0.1



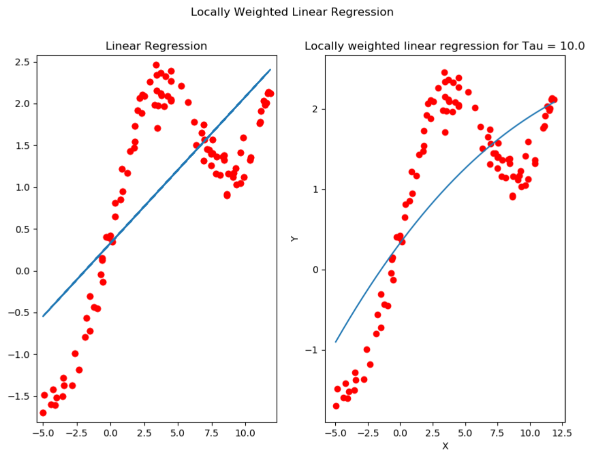
(b) Bandwidth parameter = 0.3



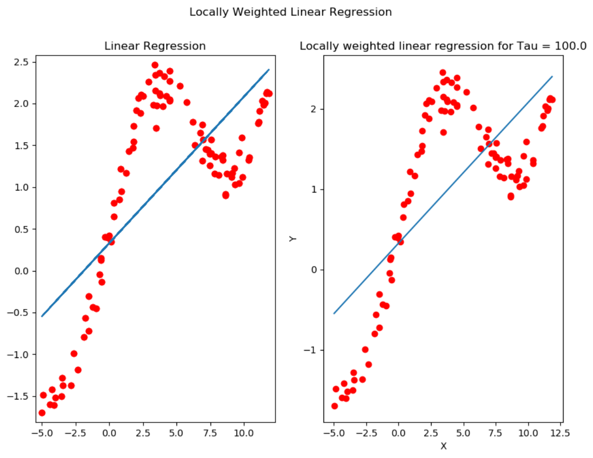
(c) Bandwidth parameter = 0.8



(d) Bandwidth parameter = 2



(e) Bandwidth parameter = 10



(f) Bandwidth parameter = 100

Figure 3: Plots for bandwidth parameter for Locally weighted linear regression

Logistic Regression

Logistic regression is a supervised classification algorithm. We use logistic regression for binary classification of a set of training examples. For the purpose of learning the model we use variants of Gradient descent and also Newton's method.

- Hypothesis function:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{18}$$

- We maximize the log likelihood of the generative model defined as:

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) \tag{19}$$

$$LL(\theta) = \sum_{i=1}^{m} log(p(y^{(i)}|x^{(i)};\theta)) \tag{20}$$

$$where\ p(y^{(i)}|x^{(i)};\theta) = (h_\theta(x^{(i)})^{y^{(i)}})(1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \tag{21}$$

- To maximize this function we use Newton's method (also called Fisher scoring)

$$\theta^{t+1} = \theta^t - H^{-1}\nabla L(\theta)|_{\theta^t} \tag{22}$$

- We also use Batch and Stochastic gradient descent to reach the maxima of likelihood function:

$$\theta^{t+1} = \theta^t + \nabla L(\theta)|_{\theta^t} \tag{23}$$

- Threshold = 1 $\times$ 10^{-10}
- Condition for convergence: $\Delta\theta < Threshold$
- Solution attained: [[0.40125316] [2.5885477] [-2.72558849]]

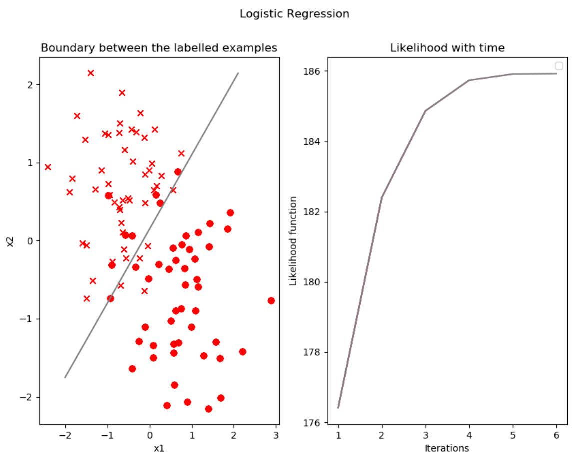
The number of iterations and time take are shown in Table 2.

Algorithm	Time	Number of iterations
Newton's Method	0.089999	9
BGD	27.12892	798
SGD	10.22399	2772

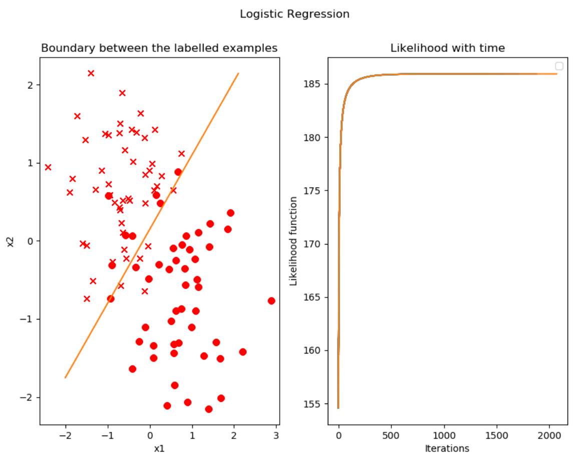
Table 2: Table showing time and iterations used for different classification approaches

Observations:

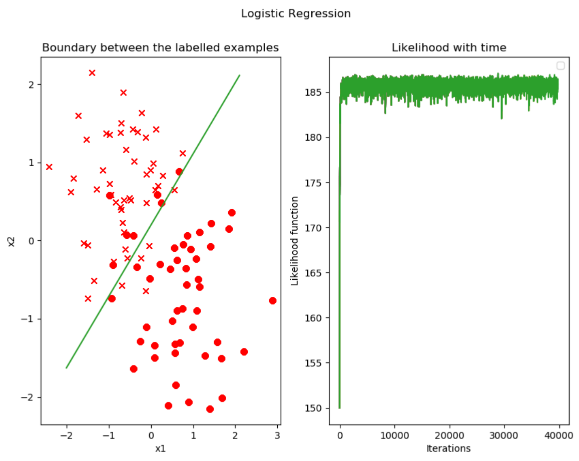
- Figure 4 shows different graphs for various approaches used to maximize likelihood of generative model
- We clearly see that for low number of training examples (100 in our case), the number of iterations as well as time is much lower by using Newton's method.



(a) Newton's method



(b) Batch Gradient Descent



(c) Stochastic Gradient Descent

Figure 4: Plots for Logistic regression using different approaches

Gaussian Discriminant Analysis

This model is a generative model for binary classification using a linear or quadratic separator (hyperbola or ellipse) for labelled training data.

- Empirically calculated metrics:

$$\phi = \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\} \tag{24}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 0\}} \tag{25}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}} \tag{26}$$

$$\Sigma_0 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 0\}(x^{(i)} - \mu_0)(x^{(i)} - \mu_0)^T}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 0\}} \tag{27}$$

$$\Sigma_1 = \frac{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}(x^{(i)} - \mu_1)(x^{(i)} - \mu_1)^T}{\sum_{i=1}^{m} \mathbb{1}\{y^{(i)} = 1\}} \tag{28}$$

- Values:

$$\phi = 0.5 \tag{29}$$

$$\mu_0 = [[98.38,\ 429.66]] \tag{30}$$

$$\mu_1 = [[137.46.\ 366.62]] \tag{31}$$

$$\Sigma_0 = [[255.3956,\ -184.3308] \tag{32}$$

$$[-184.3308,\ 1371.1044]] \tag{33}$$

$$\Sigma_1 = [[319.5684,\ 130.8348] \tag{34}$$

$$[130.8348,\ 875.3956]] \tag{35}$$

$$\Sigma = \frac{\Sigma_0 + \Sigma_1}{2} \tag{36}$$

$$= [[287.482,\ -26.748] \tag{37}$$

$$[-26.748,\ 1123.25]] \tag{38}$$

General expression for Gaussian Discriminant Analysis:

$$\frac{1}{2}(\Sigma_1^{-1} - \Sigma_0^{-1})x^T x + (\mu_0^T\Sigma_0^{-1}x - \mu_1^T\Sigma_1^{-1}x) + log(\frac{1-\phi}{\phi}) + \frac{1}{2}(\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1) - \frac{1}{2}log(\frac{|\Sigma_0|}{|\Sigma_1|}) = 0 \tag{39}$$

Which reduces to a linear expression for $\Sigma_0 = \Sigma_1$:

$$(\mu_0^T\Sigma_0^{-1}x - \mu_1^T\Sigma_1^{-1}x) + log(\frac{1-\phi}{\phi}) + \frac{1}{2}(\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1) = 0 \tag{40}$$

Observations:

- We observe that among the linear and quadratic separators, the quadratic separator has lower classification error as some Alaska fish are classified correctly (not in linear separator case)
- We also observe when we zoom out the quadratic separator is actually a hyperbola

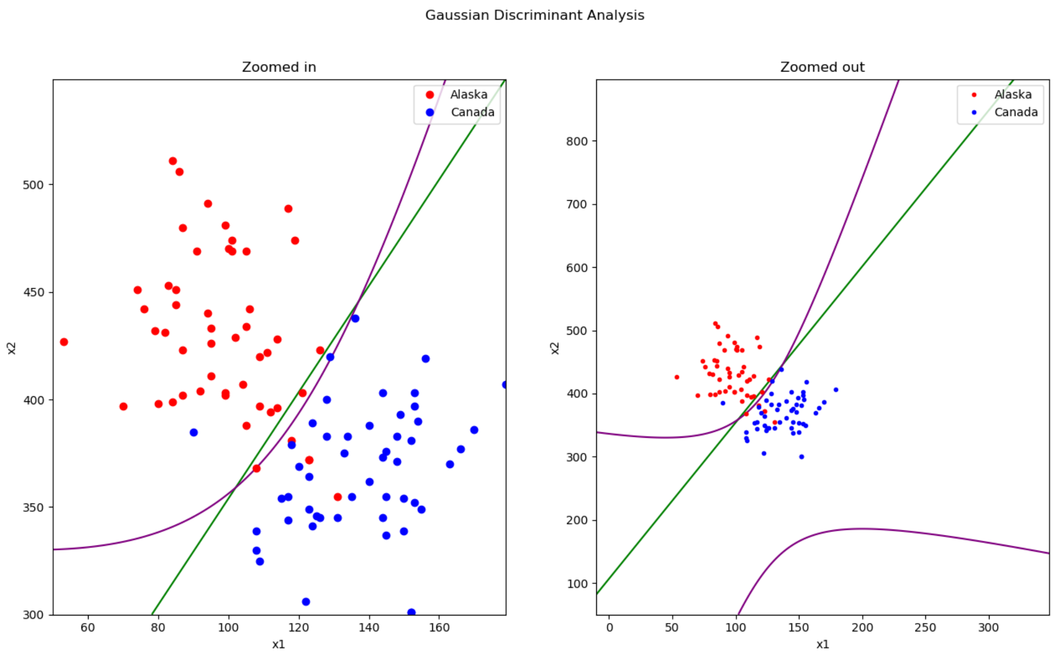


Figure 5: Gaussian Discriminant Analysis - showing linear and quadratic separators