COL774 Assignment 1

Rishabh Kumar (2021CS10103)

August 2023

1 Linear Regression(Gradient Descent)

Update Rule:

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

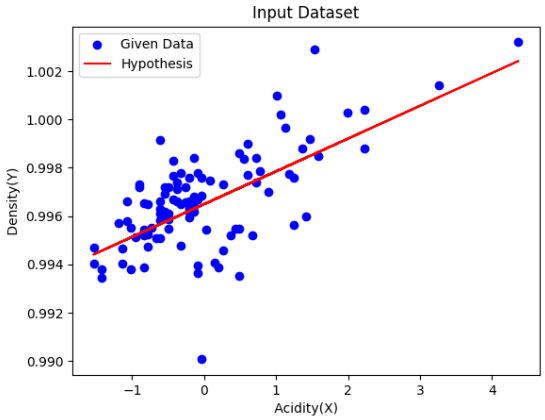
$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla J(\theta^{(t)}) \tag{2}$$

1.1 Gradient Descent

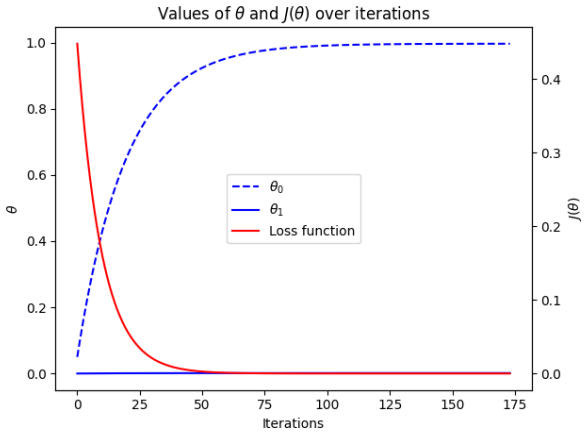
- **Learning Rate :** $\alpha = 0.1$
- **Stopping Criteria :**
  - $iterations > 1000$
  - $|J(\theta_i) - J(\theta_{i-1})| < 10^{-9}$
- **Learned Parameters :** $\theta_0 = 0.99650157, \theta_1 = 0.00135776$
- **Observations :** Smaller learning rate fits data more efficiently i.e. up to 10^{-10} but require more iterations but in our case when $\alpha = 0.1$, it takes 174 iterations. Also, 10^{-9} is still good considering as it converges in much less iterations and cost decrease is far less.

1.2 Data and Learned Hypothesis function

Plotting the given data with normalised X on a two-dimensional graph and the hypothesis function learned by the GD algorithm with conditions in the previous part.



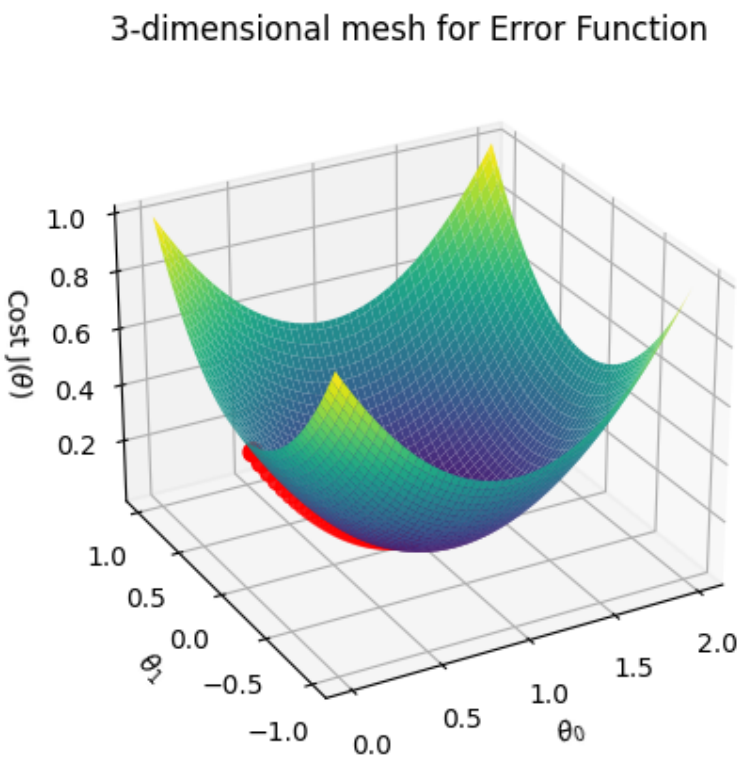
(a) Data and hypothesis function for linear regression



(b) Learning Path vs Iterations

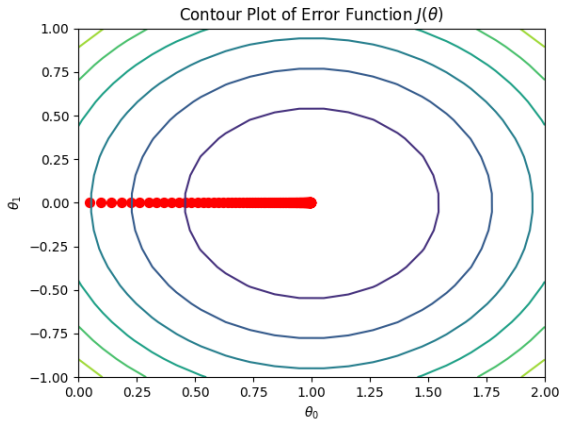
1.3 Loss vs Parameters 3D plot

The figure illustrates a 3-dimensional mesh showing the error function ($J(\theta)$) on the z-axis and the parameters in the x-y plane. Also, the red colored path displays the error value using the current set of parameters at each iteration of the gradient descent. The movement of this path can also be observed by running the code.



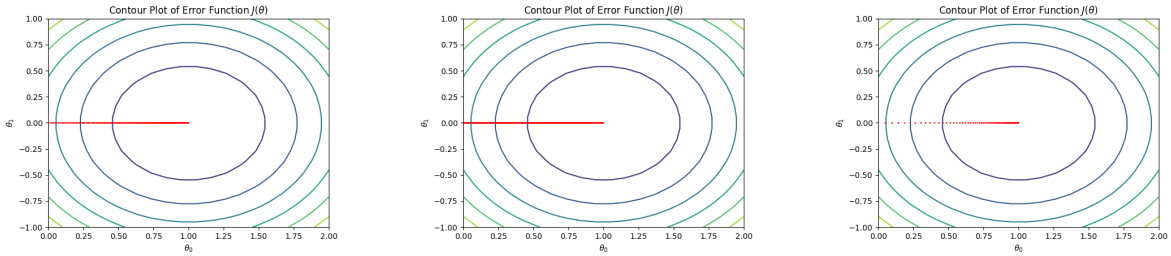
1.4 Contours of the error function

The figure is the contours of the error function at each iteration of the gradient descent. The movement of the red path can also be observed by running the code.



1.5 Contours for different step sizes

α	Number of Iterations	Final Cost	Final Theta
0.025	651	$1.2204476725495424 \times 10^{-6}$	$\theta_0 = 0.99635733, \theta_1 = 0.00135756$
0.001	13114	$2.180478087737145 \times 10^{-6}$	$\theta_0 = 0.9952211, \theta_1 = 0.00135601$
0.1	174	$1.190921476713774 \times 10^{-6}$	$\theta_0 = 0.99650157, \theta_1 = 0.00135776$



(a) $\alpha = 0.025$ (b) $\alpha = 0.001$ (c) $\alpha = 0.1$

Figure 2: Contour plots for different α values.

As can be seen in Figure 2, the different step sizes converge in a very similar way. The final parameters learned after the entire training process were very similar for all three step sizes of 0.001, 0.025 and 0.1. From the table, With the learning rate of 0.1, the loss has converged in 174 iterations, while with a step size of 0.025, it took 651 iterations. With a step size of 0.001, it takes 13114 iterations. Allowing them to converge completely, the three experiments required 174, 651 and 13114 iterations respectively to converge, implying that a larger learning rate allows the Gradient Descent algorithm to converge faster.

2 Sampling and Stochastic Gradient Descent

2.1 Sampling of Data

The following code snippet demonstrates generating sample data with the given parameters and given *samplesize*. The data includes normally distributed variables x_1 and x_2, as well as noise (ϵ) added to the dependent variable y. The relationship between the variables is defined as:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \epsilon$$

```
def sample_data(sample_size):
    np.random.seed(0)
    # Given parameters
    theta = np.array([3, 1, 2]).reshape(-1, 1)  # Reshape to make it a column vector
    # Generate x1 and x2 from normal distributions
    x1 = np.random.normal(3, 2, sample_size)
    x2 = np.random.normal(-1, 2, sample_size)
    # Generate noise for y from normal distribution
    noise = np.random.normal(0, np.sqrt(2), sample_size)
    noise = noise.reshape(-1, 1)  # Reshape to make it a column vector
    # Calculate Y using the given theta values
    X = np.column_stack((np.ones(sample_size), x1, x2))
    Y = np.dot(X, theta) + noise
    return X, Y

sample_X, sample_Y = sample_data(1000000)
```

2.2 Stochastic Gradient Descent

To implement SGD, after each epoch, the data is randomly shuffled, and then divided into batches of size r. The training update step is called with the data in each of these batches separately. For convergence, let k be a parameter which decides number of iterations over which the cost is being averaged before checking convergence criteria. Also, k is inversely proportional to batch size. As batch size decreases, noisier is the step of theta. Hence, we need to take average of larger iterations. Let us define $AD(i,k)$, as:

$$AD(i,k) = \frac{1}{k}\sum_{j=i-k}^{i} |J(\theta_j) - J(\theta_{j-k})| \tag{3}$$

Batch Size(r)	**Stopping Criteria**	**Learned Theta**	**Loss**
1	$AD(i,1000) < 10^{-9}$	θ = [[3.02706905][1.04659273][1.95858245]]	1.029930371734856
100	$AD(i,1000) < 10^{-9}$	θ = [[2.9996814][0.99886581][2.0004612]]	1.0002458139901582
10000	$AD(i,100) < 10^{-9}$	θ = [[2.99927222][0.99946337][1.99896396]]	1.000236841390263
1000000	$AD(i,1) < 10^{-9}$	θ = [[2.99429252][1.00053795][1.99862331]]	1.0002405233458718

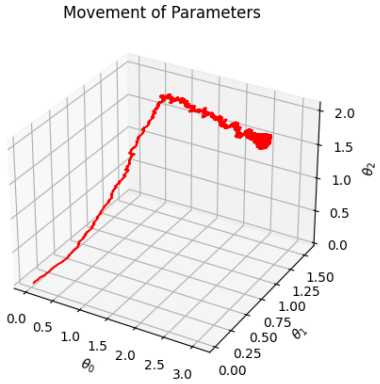
2.3 Part (c): Observations

From the table above we can clearly see that, the parameters learned with all of these batch sizes are very close and nearly equal to original theta, but are not exactly identical. Though it should be expected that the theta converges more at higher batch sizes as it is using a large number of data for the learning step but since the learning rate is small enough it is counteracting the effect. The speed of convergence and the number of iterations, however are significantly different. Larger batch size require less number of iterations but net epochs are more and hence the time taken to converge. Thus, as the batch size increases, the time to converge increases significantly.
Also, the cost generally shows a decreasing trend with increasing batch size due to reason explained above of better convergence at higher batch size. The errors on the new test data of 10,000 samples provided in the file named *q2test.csv* are summarised below:

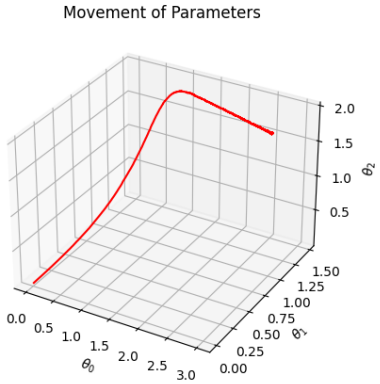
Batch Size(r)	**Loss (Trained θ)**	**Loss (Original θ)**
1	1.1895265899818308	0.9829469215
100	0.9830335727110631	0.9829469215
10000	0.9831954734041658	0.9829469215
1000000	0.9832546677127068	0.9829469215

2.4 Movement of theta for different batch sizes

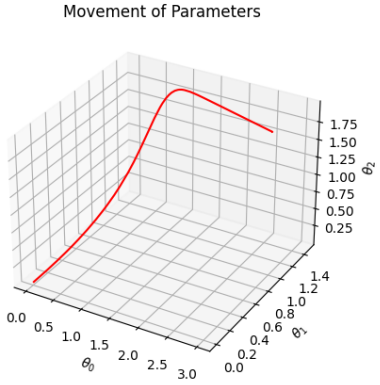
Observing the figures below, the movement of the parameter θ for a batch size of 1 is more noisy in nature. As the batch size increases, the curve becomes smoother. With a batch size of 100, some noisy nature is still visible, while batch sizes of 10,000 and 100,000 look completely smooth. This makes intuitive sense also because different batches of small numbers of samples will have much more variation than larger numbers of samples. Larger batch sizes involve averaging over more samples, leading to more uniformity. The large variation in the loss results in a large variation in the gradients, which in turn leads to the large variation in the movement of θ. In the extreme case of a batch size of 1, the loss changes after every sample, and thus, has the most variation.



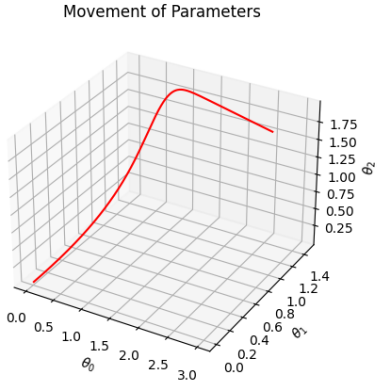
(a) Batch Size 1



(b) Batch Size 100



(c) Batch Size 10000



(d) Batch Size 100000

3 Logistic Regression

3.1 Part (a)

Using Log-likelihood loss function:

$$LL(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)}logh_\theta(x^{(i)}) + (1-y^{(i)})log(1-h_\theta(x^{(i)}))) \tag{4}$$

where Hypothesis function is:

$$h_\theta(x) = \frac{1}{1+e^{-X\theta}} \tag{5}$$

Using Newton's Method of update discussed in the class:

$$\theta := \theta - H^{-1}\nabla_\theta.l(\theta) \tag{6}$$

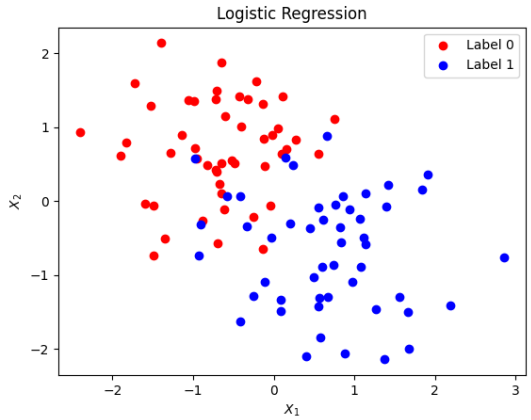
where the Hessian function is:

$$H_\theta(LL(\theta) = X^T.diag(\sigma(X.\theta)(1-\sigma(X.\theta))).X \tag{7}$$

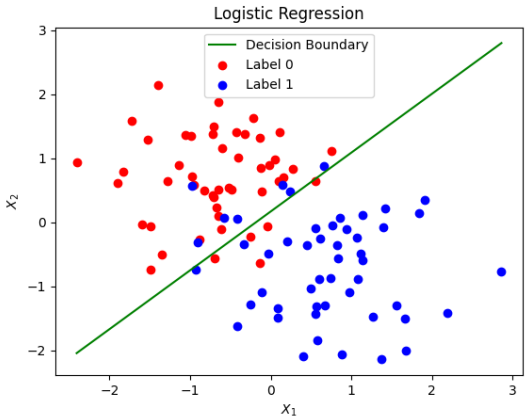
- **Convergence Criteria:** $|LL(\theta_i) - LL(\theta_{i-1})| < 10^{-6}$
- **Learned** θ = [[0.46722676] [2.55770122] [-2.7814376]]

3.2 Plot of training data and Decision Boundary

The decision boundary is derived from the equation $h_\theta(x(i)) = 0.5$, also expressed as $\theta^T X = 0$. We will use the latter formulation to implement the decision boundary. This plot has been depicted in the below Figure.



(a) Training Data



(b) Training Data with Decision Boundary

4 Gaussian Discriminant Analysis

4.1 Part (a)

Following the equations in discussed in class for finding μ_0 and μ_1. The expressions are below:

$$\begin{aligned} \mu_0 &= \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}}, \\ \mu_1 &= \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\} x^{(i)}}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}} \end{aligned} \tag{8}$$

Assuming that both the classes have the same co-variance matrix i.e. $\Sigma_0 = \Sigma_1 = \Sigma$, we have its expression as below:

$$\Sigma = \frac{\sum_{i=1}^{m} (x^{(i)} - \mu_y^{(i)})(x^{(i)} - \mu_y^{(i)})^T}{m} \tag{9}$$

where,

$$\mu_y^{(i)} = 1\left\{y^{(i)} = 0\right\} \mu_0 + 1\left\{y^{(i)} = 1\right\} \mu_1 \tag{10}$$

Using above equations, the parameters obtained on the given data are:

- $\mu_0 = [-0.755294330.68509431]$
- $\mu_1 = [0.75529433 - 0.68509431]$
- $\Sigma = \begin{bmatrix} 0.42953048 & -0.02247228 \\ -0.02247228 & 0.53064579 \end{bmatrix}$
- $\Sigma = \begin{bmatrix} 0.38158978 & -0.15486516 \\ -0.15486516 & 0.64773717 \end{bmatrix}$

4.2 Part (b)

Here(5), the points with blue dots are data points that have label 'Canada' while those with red dots have label 'Alaska'. Below is the plot for normalised data:

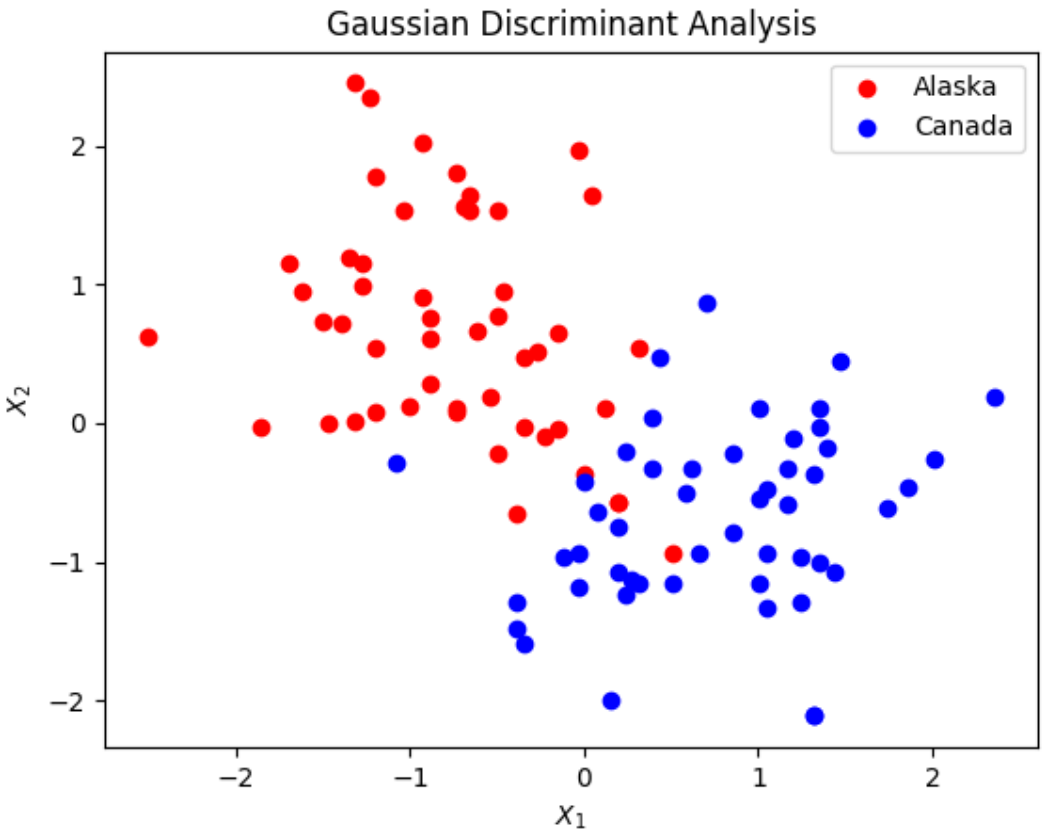


Figure 5: Normalised data for GDA

4.3 Part (c)

As discussed in class, at the decision boundary, $\theta^T X = \mathrm{h} = 0$. Thus, the decision boundary can be described in terms of μ_0, μ_1 and Σ as in below equation, which has been implemented directly in the code.

$$-(\mu_1 - \mu_0)^T \Sigma^{-1} x + \frac{1}{2}(\mu_1^T \Sigma^{-1} \mu_1 - \mu_0^T \Sigma^{-1} \mu_0) + \log \frac{(1-\phi)}{\phi} = 0 \tag{11}$$

where,

$$\phi = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}}{m} \tag{12}$$

```python
def plot_linear():
    x1_vals = np.linspace(-2, 2, 2)
    # Calculate x2 values using the h function
    coefficient_term = -1 * np.dot(np.transpose(mu_1 - mu_0), sigma_inv)
    constant_term_1 = np.dot(np.dot(np.transpose(mu_1), sigma_inv), mu_1)
    constant_term_2 = np.dot(np.dot(np.transpose(mu_0), sigma_inv), mu_0)
    constant_term_3 = np.log((1 - phi) / phi)
    constant_term = (constant_term_1 - constant_term_2) / 2 + constant_term_3
    x2_vals = []
    for x1 in x1_vals:
        # Calculate x2 from h value (inverse of h function)
        x2 = -1*(constant_term+coefficient_term[0]*x1) / coefficient_term[1]
        x2_vals.append(x2)
    x2_vals = np.array(x2_vals)
    return x1_vals, x2_vals
x1_vals, x2_vals = plot_linear()
plt.plot(x1_vals, x2_vals, color='green')
```

Below is the plot (6) of the decision boundary found by using parameters we get in Part A.

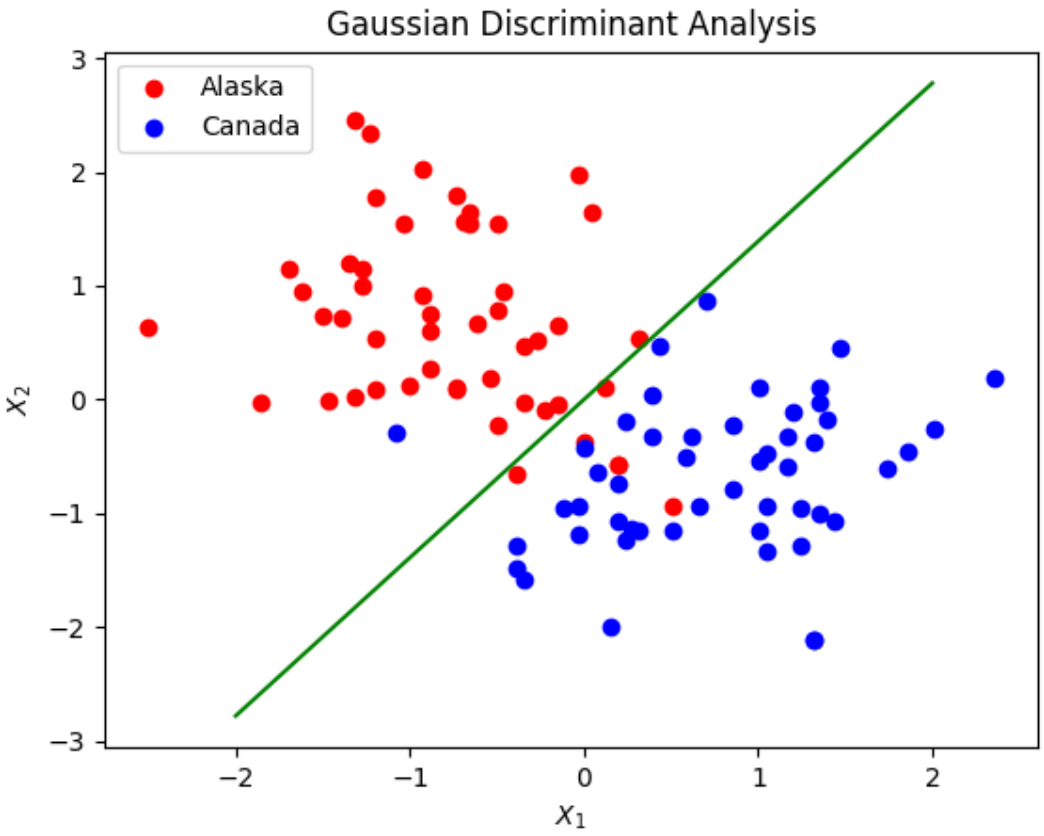


Figure 6: Linear Decision boundary for GDA

4.4 Part (d)

In this part we will calculate parameters for general GDA, where $\Sigma_0 \neq \Sigma_1$ and calculated using expressions below:

$$\Sigma_0 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}(x^{(i)} - \mu_y^{(i)})(x^{(i)} - \mu_y^{(i)})^T}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 0\right\}} \tag{13}$$

$$\Sigma_1 = \frac{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}(x^{(i)} - \mu_y^{(i)})(x^{(i)} - \mu_y^{(i)})^T}{\sum_{i=1}^{m} 1\left\{y^{(i)} = 1\right\}} \tag{14}$$

where $\mu_y^{(i)}$ is the same as that used in eq. 10 and also for finding μ_0 and μ_1 we can use same equation as eq. 8. Using above equations, the parameters obtained on the given data are:

- $\mu_0 = [-0.755294330.68509431]$
- $\mu_1 = [0.75529433 - 0.68509431]$
- $\Sigma_0 = \begin{bmatrix} 0.38158978 & -0.15486516 \\ -0.15486516 & 0.64773717 \end{bmatrix}$
- $\Sigma_1 = \begin{bmatrix} 0.47747117 & 0.1099206 \\ 0.1099206 & 0.41355441 \end{bmatrix}$

4.5 Part (e)

In this case, we get a parabolic decision boundary which can be described in terms of μ_0, μ_1, Σ_0 and Σ_0 as in below equation:

$$\frac{1}{2}(x^T(\Sigma_1^{-1} - \Sigma_0^{-1})x) - (\mu_1^T\Sigma_1^{-1} - \mu_0^T\Sigma_0^{-1})x + \frac{1}{2}(\mu_1^T\Sigma_1^{-1}\mu_1 - \mu_0^T\Sigma_0^{-1}\mu_0) + \log(\frac{(1-\phi|\Sigma_1|^{\frac{1}{2}})}{\phi|\Sigma_0|^{\frac{1}{2}}}) = 0 \quad (15)$$

The direct implementation of above quadratic equation in the code, can be seen in the code snippet below. And fig. 7 illustrates the plot.

```
def plot_quadratic():
    # squareroot of determinant of cov-matrices
    d_1 = np.sqrt(np.linalg.det(sigma_1))
    d_0 = np.sqrt(np.linalg.det(sigma_0))
    # Evaluating the final expression h
    coefficient_term_1 = 0.5*(sigma_1_inv - sigma_0_inv)
    coefficient_term_2 = np.dot(mu_1.T, sigma_1_inv) - np.dot(mu_0.T, sigma_0_inv)
    constant_term = 0.5*(np.dot(np.dot(mu_1.T, sigma_1_inv), mu_1) - np.dot(np.dot(mu_0.
    T, sigma_0_inv), mu_0)) + np.log((1-phi*d_1)/phi*d_0)
    x1_vals = np.linspace(-3, 3, 200)
    x2_vals = np.linspace(-3, 3, 200)
    x1, x2 = np.meshgrid(x1_vals, x2_vals)
    term_1 = (x1**2)*coefficient_term_1[0][0] + x1*x2*(coefficient_term_1[0][1] +
    coefficient_term_1[1][0]) + (x2**2)*coefficient_term_1[1][1]
    term_2 = coefficient_term_2[0]*x1 + coefficient_term_2[1]*x2
    # h = 0
    h = term_1 - term_2 + constant_term
    return plt.contour(x1, x2, h, levels=[0], colors='orange')
```

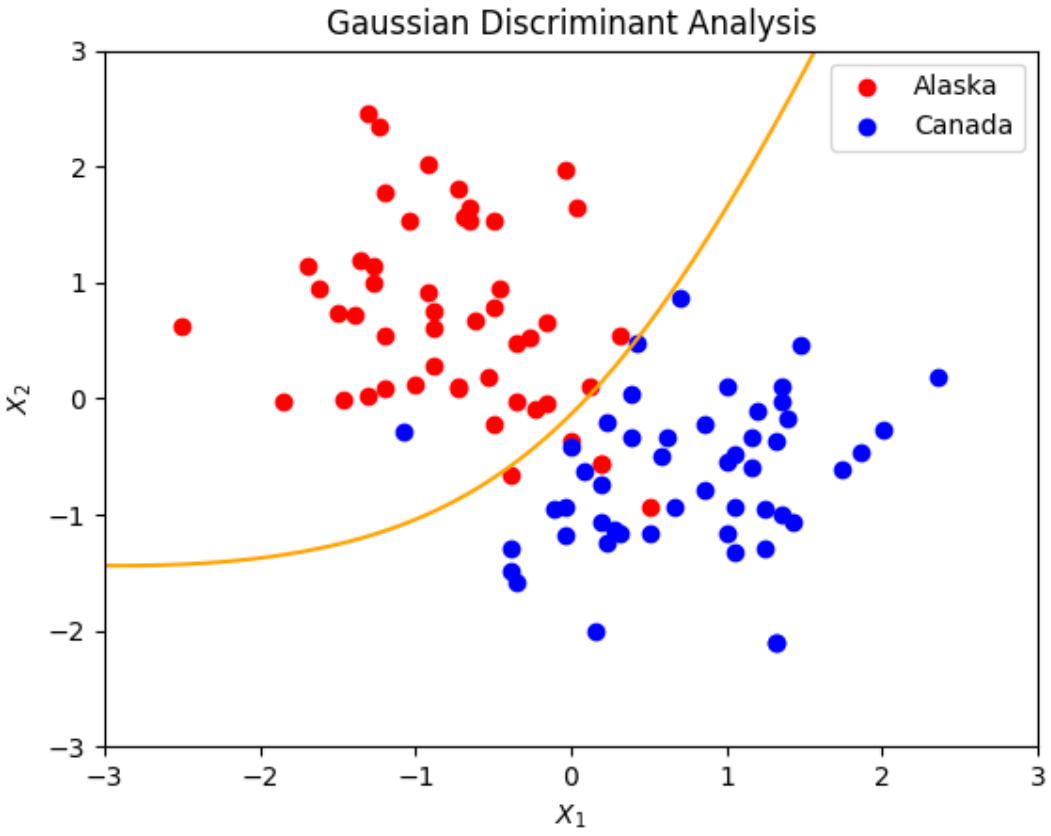


Figure 7: Quadratic Decision Boundary for GDA

4.6 Part (f) : Analysis of Decision Boundaries

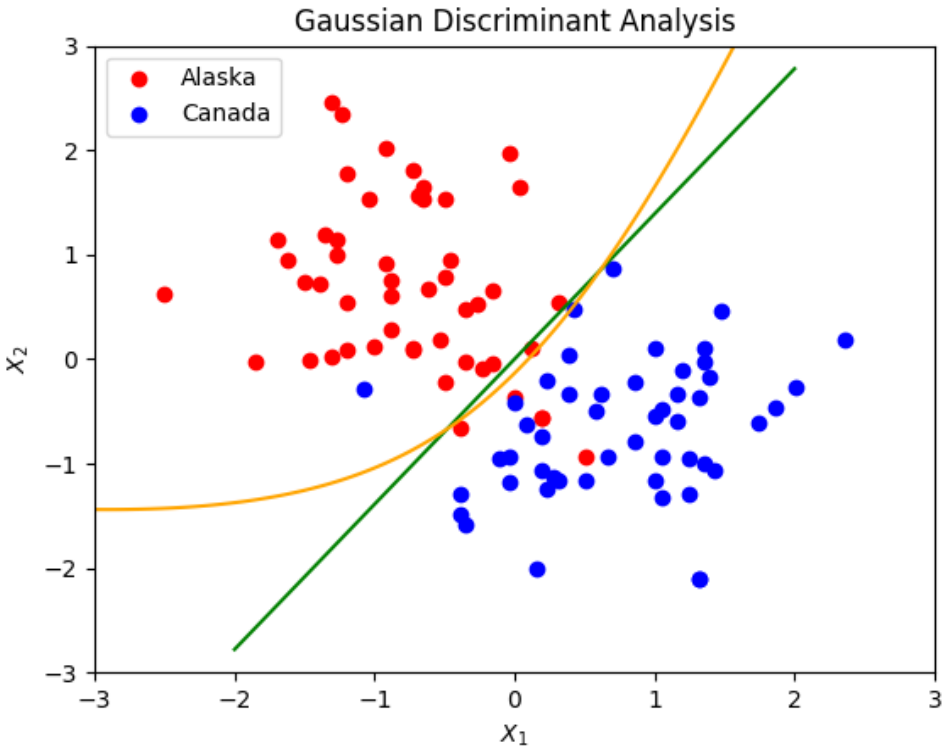


Figure 8: Both Linear and Quadratic Boundaries together

- In figure 8, we can observe a small gap between the two decision boundaries. Within this gap, approximately 2-3 samples are labeled as "Alaska". Comparing this to figure with linear decision boundary, it becomes evident that the quadratic decision boundary correctly classifies these few samples, which were previously misclassified by the linear boundary.
- Regarding the curvature of the decision boundaries, it's notable that the curvature tends to point towards the center of the cluster with the smaller magnitude of $|\Sigma_i|$ or, equivalently, the larger magnitude of $|\Sigma_i^{-1}|$. Here in our case, $|\Sigma_0| < |\Sigma_1|$, the boundary curves towards the cluster labeled as 0, which corresponds to the Alaska cluster.
- Intuitively, the magnitude of curvature of the boundary is inversely proportional to the absolute difference between the respective $|\Sigma_i^{-1}|$ values. When both $|\Sigma_i^{-1}|$ values are the same, the curvature is expected to increase infinitely, which is precisely what occurs when the linear boundary is obtained.