ASSIGNMENT – 1

1. LINEAR REGRESSION BY GRADIENT DESCENT

First of all, imported necessary libraries - NumPy, pandas, os and sys. Used pandas, sys and os to read the training as well as test data. Converted the data to arrays. Normalized the x's of data with mean and standard deviation mentioned in the assignment pdf. Initialized the various parameters – number of examples, θs, learning rate (η) and created a variable prev_cost to store the value of cost function J(θ). Implemented a while loop, within which the whole algorithm of Gradient Descent is implemented. Created a variable h_theta whose value is to be calculated by the formula,

$$h(\theta) = \theta_0 + x(i) * \theta_1$$

as taught in the class.

Then, created a temp variable to store the difference of h(θ) and y(i) for each iteration.
Then, used the following formula to calculate the cost function J(θ),

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(\theta) - y(i))^2$$

To get the updation equations of θs, I differentiated the J(θ) equation with respect to each of the θs independently and equated it to zero.
On following the above method, I obtained the following equations for updation,

$$\Theta_0 = \Theta_0 - \eta * [\tfrac{1}{m}\textstyle\sum_{i=1}^{m} h(\theta) - y(i)]$$

$$\Theta_1 = \Theta_1 - \eta * [\tfrac{1}{m}\textstyle\sum_{i=1}^{m}\{h(\theta) - y(i)\} * \{x(i)\}]$$

Then, implemented a stopping criterion by taking the absolute of difference of current cost function value and previous value of the same.
The **stopping criterion** used is that if the absolute value of difference of current cost function (variable – cost_function) and previous value of the cost function (variable – prev_cost) is less than $\mathbf{10^{-9}}$, then, the loop will break.
Then, a variable y_test is created to store the output of test data x_test. Then, I converted y_test to a DataFrame using pandas so that the output can be obtained as a .txt file.

(a) (i) *Stopping Criteria*

|cost_function – prev_cost| < 0.000000001

(ii) *Learning Rate* = **0.1**

(iii) *Final set of parameters obtained*

$\boldsymbol{\theta_0 = 0.99667481}$

$\boldsymbol{\theta_1 = 0.00127942}$

The following plots were obtained:

(b) Plot of hypothesis function learned by the algorithm.

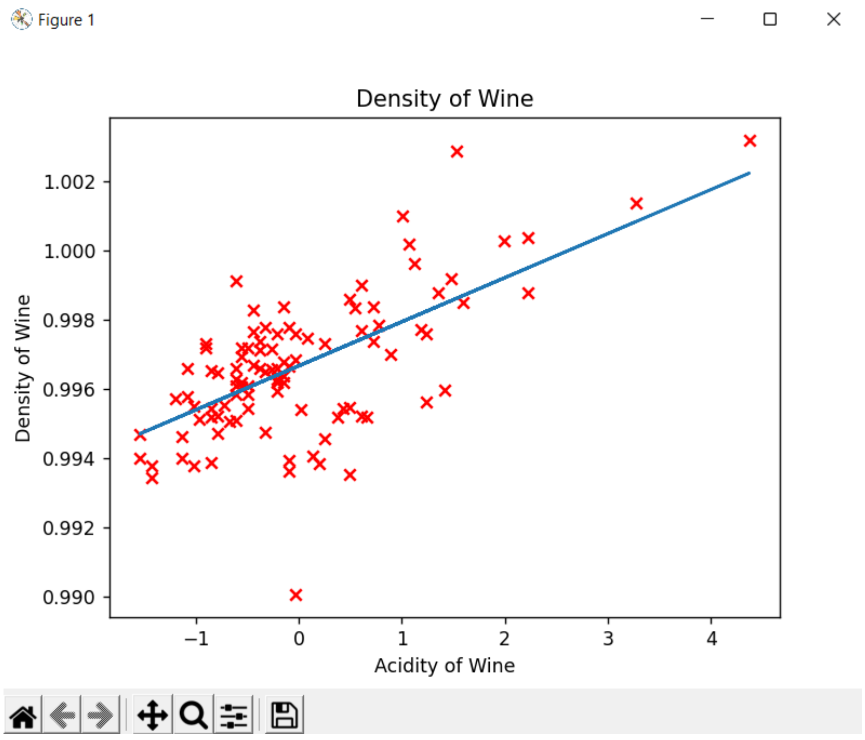


Figure 1: Linear Regression

(c) 3-dimensional mesh showing the error function

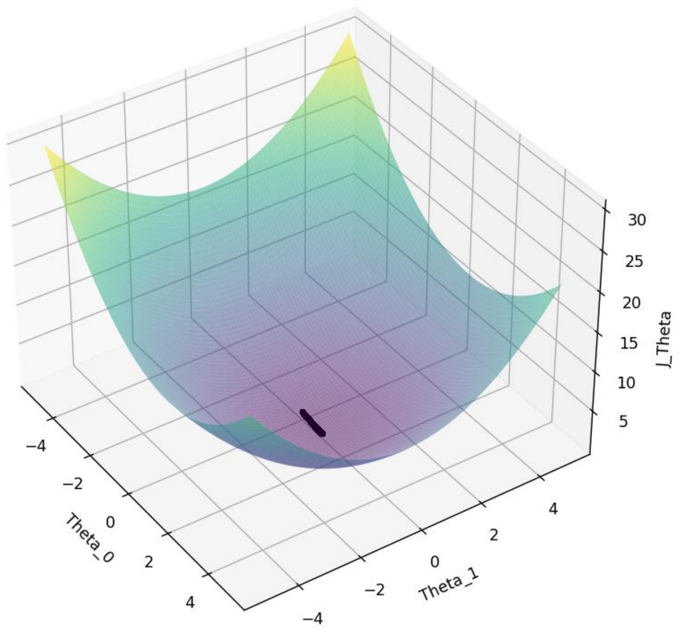


Figure 2: 3-d Mesh showing the Error Function

(d) Contour of the Error Function when η = 1.

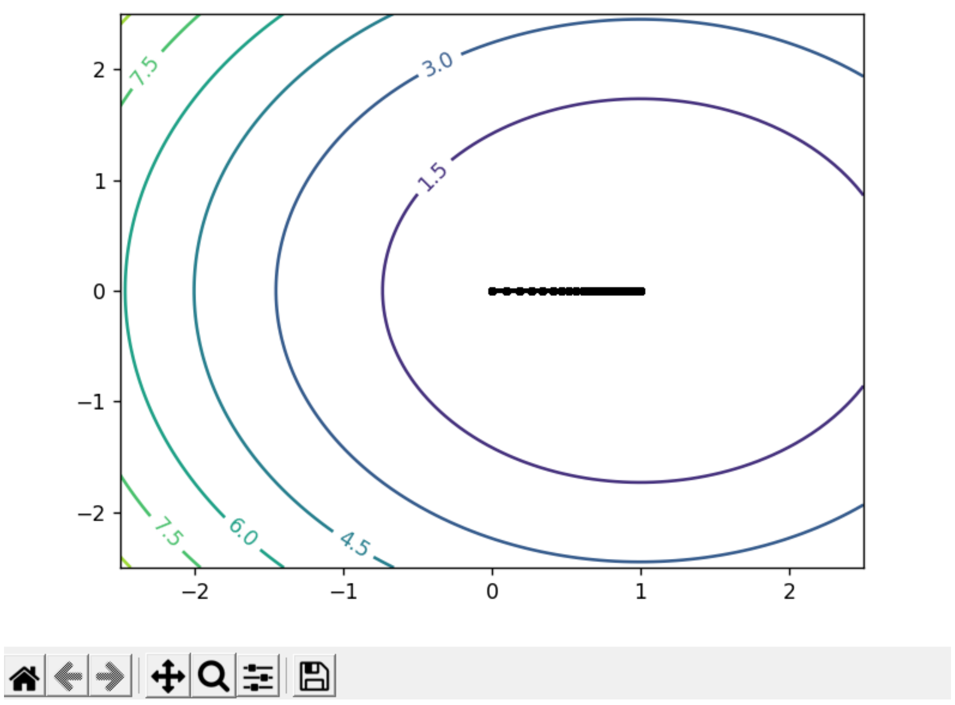


Figure 3: Contour when η = 0.1

(e) Contours of the Error function when η = 0.025 and η = 0.001

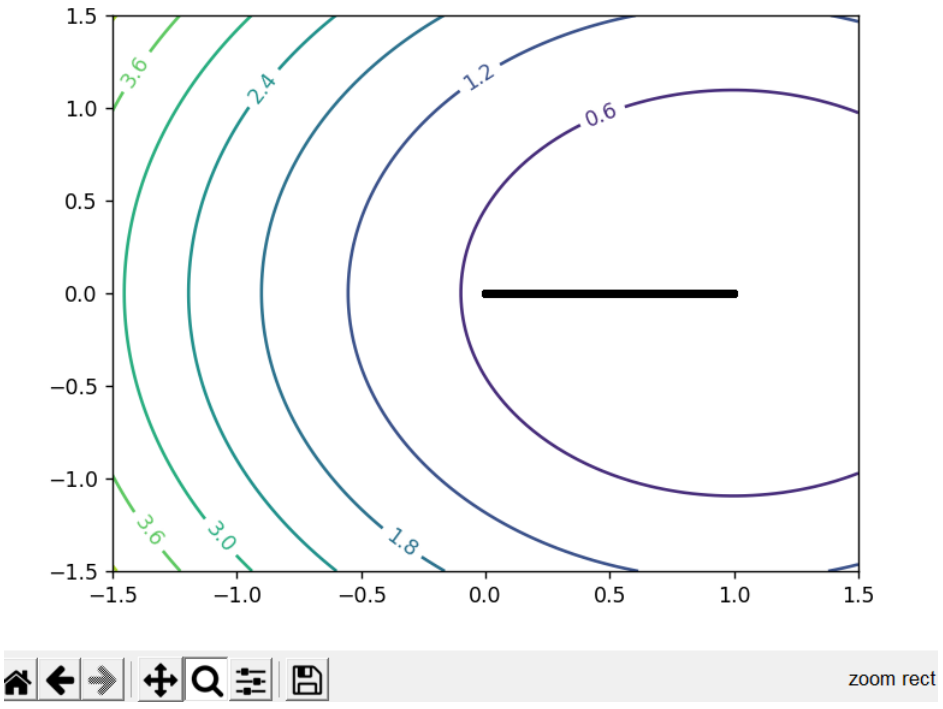


Figure 4: Contour when η = 0.025

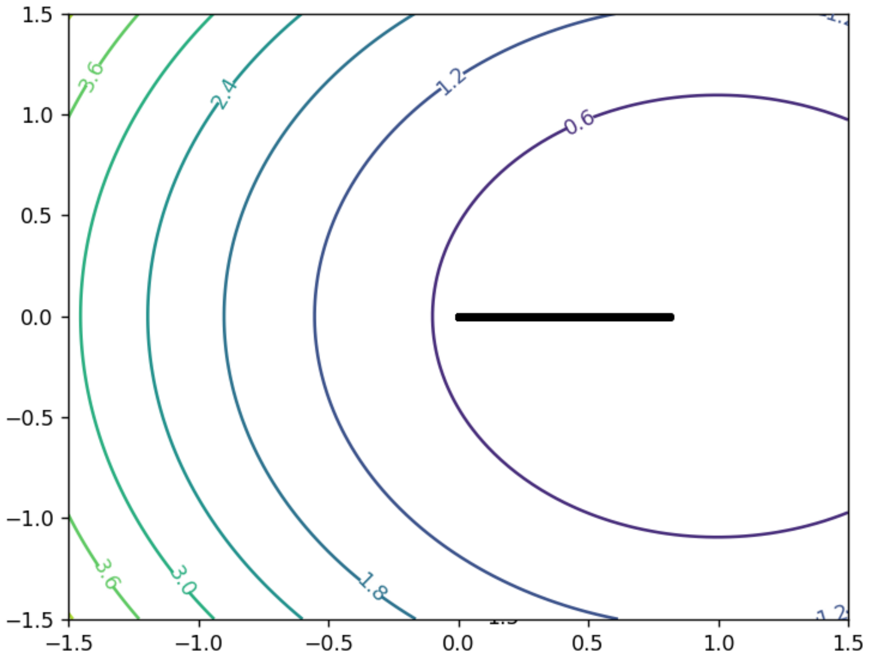


Figure 5: Contour when η = 0.001

Observations:

(i) Time taken by the gradient descent algorithm to converge and number of iterations taken when η is 0.1 = **0.03124427** and **90**

(ii) Time taken by the gradient descent algorithm to converge and number of iterations taken when η is 0.025 = **0.0624899** and **339**

(iii) Time taken by the gradient descent algorithm to converge and number of iterations taken when η is 0.001 = **1.68709826** and **6903**

From above observations, it is clear that as we keep decreasing the learning rate, the amount of time taken to converge the cost function as per our stopping criteria keeps increasing and the number of iterations it takes to converge also increases significantly.

2. SAMPLING AND STOCHASTIC GRADIENT DESCENT

First of all, imported necessary libraries - NumPy, pandas, os and sys. Used pandas, sys and os to read the training as well as test data.
According to original hypothesis, the parameters are

$$\boldsymbol{\theta_0 = 3}$$
$$\boldsymbol{\theta_1 = 1}$$
$$\boldsymbol{\theta_2 = 2}$$

Stored the above values in variables Th_0, Th_1 and Th_2.

Generated two normal distributions as asked in the question using the parameters specified. Also generated a normal distribution for noise, Epsilon.

Created a variable y to store the function value as follows:

$$\boldsymbol{y = \theta_0 + \theta_1 * x1 + \theta_2 * x2 + \varepsilon}$$

where, θ_0, θ_1, and θ_2 are the parameters, x1 and x2 are the columns of the input data and ε is the noise added.
Then, defined a function named **"sgd"**, that takes two arguments batch size and stopping criteria.
sgd encloses the whole algorithm for Stochastic Gradient Descent.
Initialized some variables to store the values of different θs, number of iterations.
Initialized three lists t_0, t_1 and t_2 to store the theta values.
Initialized the while True loop, within which converted x data to array.
Implemented a for loop to create batches of desired sizes.
Stored y data in an array as well.
Created a variable h_theta whose value is to be calculated by the formula,

$$\boldsymbol{h(\theta) = \theta_0 + x1(i) * \theta_1 + x2(i) * \theta_2}$$

as taught in the class.
Then, created a temp variable to store the difference of h(θ) and y(i) for each iteration.
Then, used the following formula to calculate the cost function J(θ),

$$\boldsymbol{J(\theta) = \frac{1}{2(b)} \sum_{i=1}^{b} (h(\theta) - y(i))^2}$$

Where, **"b"** is the batch size.

By following the video mentioned in the question, to converge the values of θs in a reasonable amount of time, the learning rate needs to be dynamic. By trial and error, I reached to a function of learning rate that was able to achieve this.
Appending the lists t_0, t_1 and t_2 created before with the values of thetas.
Then, updated the values of θs as follows:

$$\boldsymbol{\Theta_0 = \Theta_0 - \eta * [\tfrac{1}{b} \textstyle\sum_{i=1}^{b} h(\theta) - y(i)]}$$
$$\boldsymbol{\Theta_1 = \Theta_1 - \eta * [\tfrac{1}{b} \textstyle\sum_{i=1}^{b} \{h(\theta) - y(i)\} * \{x1(i)\}]}$$
$$\boldsymbol{\Theta_2 = \Theta_2 - \eta * [\tfrac{1}{b} \textstyle\sum_{i=1}^{b} \{h(\theta) - y(i)\} * \{x2(i)\}]}$$

Used a small if loop just to print the values of θs in case they don't converge within 10 seconds, just to check that the program is running correctly and θs on their way to convergence.

As far as the convergence criteria is concerned,
There are 3 parameters, each of which must converge up to some point so as to get the best predictions possible.
prev_theta_i is the previous value of ith θ and theta_i is the current value of ith θ. The exact conditions that are used for convergence are mentioned below.

|prev_theta_0 - theta_0| < fu*b
|prev_theta_1 - theta_1| < fu*b
|prev_theta_2 - theta_2| < fu*b

At the end of the if loop, breaking out and then updating the values of all previous theta variables.
Plotted each value of θ on a 3d plot where axes are θ_0, θ_1 and θ_2.
Then, out of the loop, calling the function **sgd** for different batch sizes and stopping criteria as asked in the question.

(a) Implemented in the code.
(b) Implemented in the code.
(c) For different values of batch sizes, the values of θs converge to almost the same values.

For batch size 1,

- The values of theta converge to:
  theta_0, theta_1, theta_2 = (2.943891, 1.050185, 2.067133)
- The number of Iterations: **1870**
- Time taken = 0.03124070167541504 seconds
- Difference in parameters from original hypothesis function:
  (0.056108, 0.050185, 0.067133)

For batch size 100,

- The values of theta converge to:
  theta_0, theta_1, theta_2 = (2.994627, 1.010429, 2.005198)
- The number of Iterations: **2521**
- Time taken = 0.22620272636413574 seconds
- Difference in parameters from original hypothesis function:
  (0.005372, 0.010429, 0.005198)

For batch size 10000,

- The values of theta converge to:
  theta_0, theta_1, theta_2 = (2.991804, 1.003077, 2.000592)
- The number of Iterations: **2428**
- Time taken = 19.250375032424927 seconds
- Difference in parameters from original hypothesis function:
  (0.008195, 0.003077, 0.000592)

For batch size 1000000,

- The values of theta converge to:
  theta_0, theta_1, theta_2 = (2.738614, 1.058141, 1.982382)
- The number of Iterations: **414**
- Time taken = 397.93817377090454 seconds
- Difference in parameters from original hypothesis function:
  (0.261385, 0.058141, 0.017617)

As evident from the observations stated above the time taken by the algorithm for different batch sizes varies. For batch size = 1, it just takes milliseconds to get the result, for batch size = 100, it takes a fraction of a second, for batch size = 10000, it takes around 20 seconds and for batch size 1000000, it takes a few minutes.
As the batch size increases, the number of iterations taken decrease.

On implementing the above algorithm on 10,000 samples provided in the dataset, the RMS error obtained for different batch sizes are as follows:

(1) Batch Size = 1
The error in prediction of y using learned model: **1.421905**

(2) Batch Size = 100
The error in prediction of y using learned model: **1.406773**

(3) Batch Size = 10000
The error in prediction of y using learned model: **1.402439**

(4) Batch Size = 1000000
The error in prediction of y using learned model: **1.541259**

(d) The 3d graphs that were required to plot:

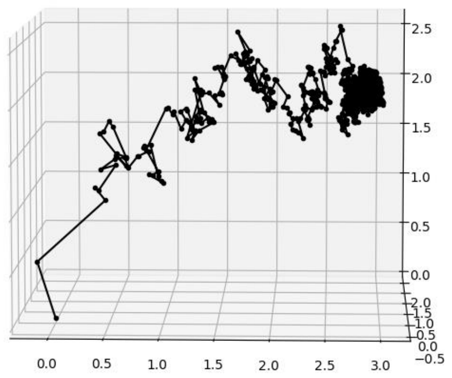


Figure 6: Plot for Batch Size = 1

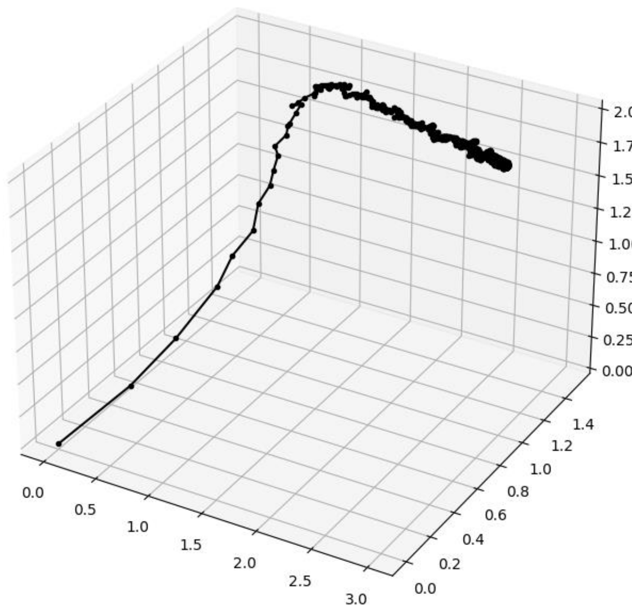


Figure 7: Plot for Batch Size = 100

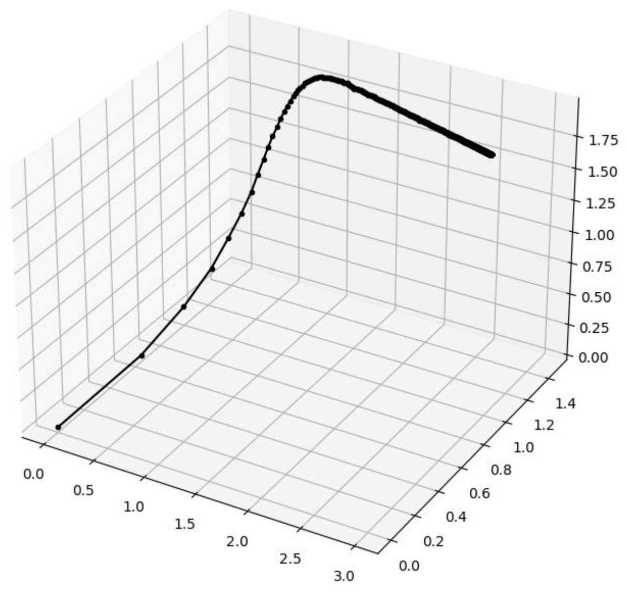


Figure 8: Plot for Batch Size = 10000

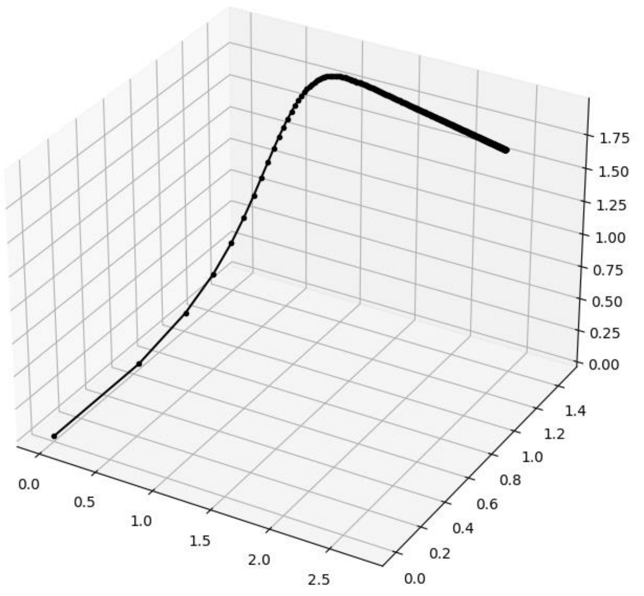


Figure 9: Plot for Batch Size = 1000000

3. LOGISTIC REGRESSION

First of all, imported necessary libraries - NumPy, pandas, os and sys. Used pandas, sys and os to read the training as well as test data. Set the learning rate $\eta = 0.001$ as provided in the question. Created an array for storing the values of parameters θ_0, θ_1 and θ_2 and initialized all of them by value (0.0). Then, created an array each for the columns of x data provided namely x1 and x2. Also, created an array x0 of all ones with the length same as x1. Stored all the 3 arrays (x0, x1 and x2) in a new array X. Created an array y to store the column vector from y data. Initialized a variable prev_cost to zero.
Implemented a while True loop, within which the whole algorithm of Logistic Regression is implemented.
Created a variable "z" whose value is to be calculated as follows:

$$\boldsymbol{z = \theta_0 + \theta_1 * x1 + \theta_2 * x2}$$

Where, θ_0, θ_1 and θ_2 are all vectors.
The hypothesis function for the algorithm is given by

$$\boldsymbol{h(z) = \frac{1}{1 + e^{-z}}}$$

which basically is a sigmoid function.
The log likelihood function for Logistic Regression as stated in the question is:

$$L(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

Implemented the above equation using NumPy as follows:
log_likelihood = np.multiply(np.log(h_theta), y) + np.multiply(np.log(1-h_theta),1-y)
Calculated the cost function by summing the above equation over all m's.
The maximization of log likelihood is done by differentiating it. Let del_J be the maximized value.
Mathematically,

del_J = Matrix multiplication of X and (y(i) – h(z)).

Calculated the Hessian as explained in class by using the formula

$$\mathbf{H = X^T D X}$$

Where D is a diagonal matrix obtained by multiplying h(z) with (1 – h(z)).
Calculated H^{-1} once H was obtained.
Used the following formula to obtain the updated parameter θs:

$$\theta := \theta - H^{-1} \nabla_\theta \ell(\theta).$$

Where l(θ) is same as del_J.
Then, used the same stopping criterion as question 1,

$$\mathbf{prev\_cost - cost < 10^{-9}}$$

to obtain the final values of all parameters θ_0, θ_1 and θ_2.

(a) The thetas obtained = **[$\theta_0 = 0.401253$, $\theta_1 = 2.588547$, $\theta_2 = -2.725588$]**

(b) The Logistic Regression Plot

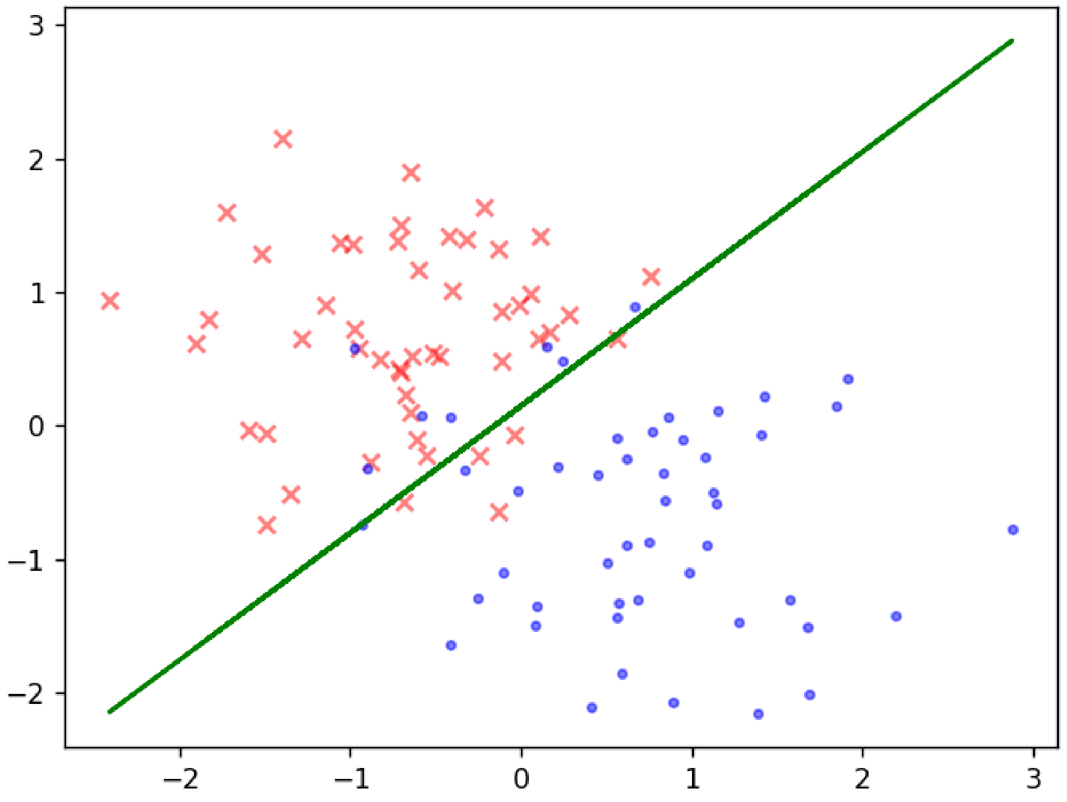


Figure 10: Logistic Regression

4. GAUSSIAN DISCRIMINANT ANALYSIS

First of all, imported necessary libraries - NumPy, SciPy, pandas, os and sys. Used pandas, sys and os to read the training as well as test data. Converted the data to arrays.
As the data is in the form of two classes, calculated the probability (ϕ) of each class by counting repetitions of keyword (Alaska and Canada) and dividing it by total number of keywords.
Created an indicator function as it will be used in further calculations.
Calculated the mean and covariance matrices for two given classes as follows:

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

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

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

$$\Sigma = \frac{1}{m}\Sigma_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}}).(x^{(i)} - \mu_{y^{(i)}})^T$$

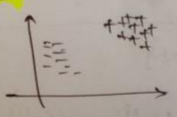
These formulas were also provided by sir in the class.
Also, created two different covariance matrices Σ_1 and Σ_2 as well.
When these two matrices are the same, the separator becomes linear.
That matrix is represented by Σ above.

(a) The values of parameters obtained

μ_0 = [0.75529433   -0.68509431]
μ_1 = [-0.75529433   0.68509431]
Σ = [1.010101   -0.545373]
[-0.545373   1.010101]

Executed a for loop to plot the scatter points as per dataset.
For plotting the decision boundary, I followed the steps provided by sir in the class, the photos of which I've attached below, to get the equation of line.

GDA.



$y^{(i)} \sim \text{Bernoulli}(\phi)$

$x^{(i)} | y^{(i)} \sim N(\mu_{y^{(i)}}, \Sigma_{y^{(i)}})$

$\theta = \cdot$

$$LL(\theta) = \ln\left[\prod_{i=1}^{m} P(y^{(i)}|\phi)\, P(x^{(i)}|y^{(i)};\phi\right]$$

$$\ln\left[\frac{P(y=1|x;\theta)}{P(y=0|x;\theta)}\right] \text{———} (1)$$

$$= \ln\left[\frac{\frac{P(y=1;\phi)\,P(x|y=1;\theta)}{\cancel{P(x;\theta)}}}{\frac{P(y=0;\phi)\,P(x|y=0;\theta)}{\cancel{P(x;\theta)}}}\right]$$

$$= \ln\left[\frac{\phi \frac{1}{(2\pi)^{n/2}(\Sigma_1)^{1/2}} e^{-\left[\frac{(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)}{2}\right]}}{(1-\phi)\frac{1}{(2\pi)^{n/2}(\Sigma_0)^{1/2}} e^{-\left[\frac{(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)}{2}\right]}}\right]$$

For $x(i)$,

$$g_i(x) = \ln\left[\left(\frac{\phi}{1-\phi}\right)\left(\frac{|\Sigma_0|}{|\Sigma_1|}\right)^{1/2}\right] - \frac{1}{2}\left[(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)\right]$$

For our case,

$\phi = 1/2$

$\therefore 1-\phi = \phi = 1/2.$

$\therefore \frac{\phi}{1-\phi} = 1$

Also, $\Sigma_0 = \Sigma_1$,

$\therefore \frac{|\Sigma_0|}{|\Sigma_1|} = 1$

$\therefore$ The constant term,

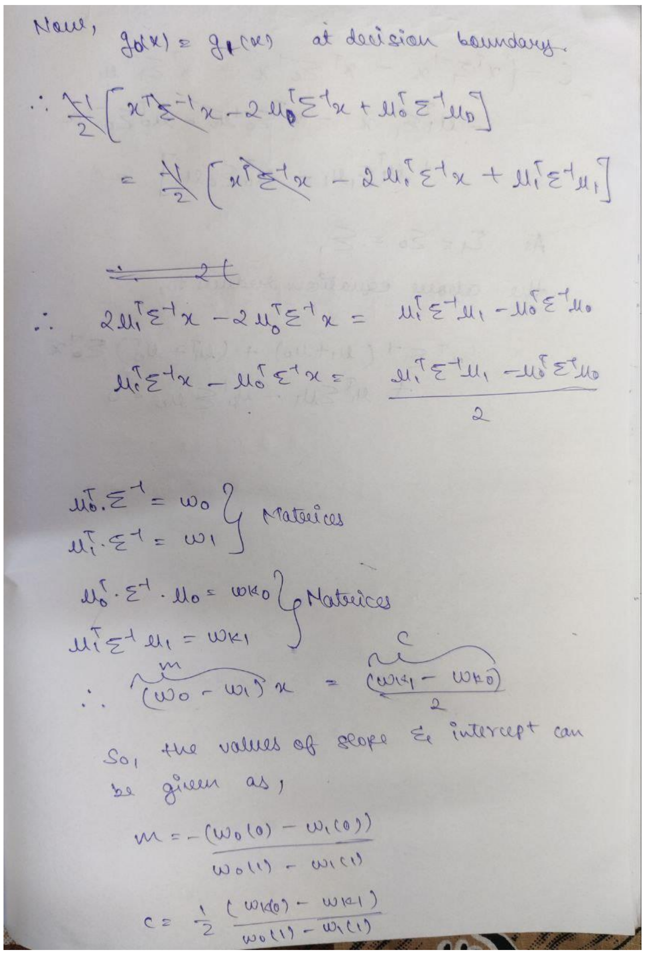
$$\ln\left[\left(\frac{\phi}{1-\phi}\right)\left(\frac{|\Sigma_0|}{|\Sigma_1|}\right)^{1/2}\right] = 0$$

$$\therefore g_i(x) = \frac{-1}{2}\left[(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)\right]$$

$$\therefore g_i(x) = \frac{-1}{2}\left[x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_i - \mu_i^T\Sigma^{-1}x + \mu_i^T\Sigma^{-1}\mu_i\right]$$

But, as $\mu_i^T\Sigma^{-1}x$ & $x^T\Sigma^{-1}\mu_i$ are both scalars, they'd be same as they are transpose of each other.

$$\therefore g_i(x) = \frac{-1}{2}\left[x^T\Sigma^{-1}x - 2\mu_i^T\Sigma^{-1}x + \mu_i^T\Sigma^{-1}\mu_i\right]$$



Now, $g_0(x) = g_1(x)$ at decision boundary.

$$\therefore \cancel{\frac{-1}{2}}\left[\cancel{x^T\Sigma^{-1}x} - 2\mu_0^T\Sigma^{-1}x + \mu_0^T\Sigma^{-1}\mu_0\right]$$

$$= \cancel{\frac{-1}{2}}\left[\cancel{x^T\Sigma^{-1}x} - 2\mu_1^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}\mu_1\right]$$

~~∴ 2(~~

$$\therefore \quad 2\mu_1^T\Sigma^{-1}x - 2\mu_0^T\Sigma^{-1}x = \mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0$$

$$\mu_1^T\Sigma^{-1}x - \mu_0^T\Sigma^{-1}x = \frac{\mu_1^T\Sigma^{-1}\mu_1 - \mu_0^T\Sigma^{-1}\mu_0}{2}$$

$$\left.\begin{aligned} \mu_0^T.\Sigma^{-1} &= w_0 \\ \mu_1^T.\Sigma^{-1} &= w_1 \end{aligned}\right\} \text{Matrices}$$

$$\left.\begin{aligned} \mu_0^T.\Sigma^{-1}.\mu_0 &= wk_0 \\ \mu_1^T\Sigma^{-1}\mu_1 &= wk_1 \end{aligned}\right\} \text{Matrices}$$

$$\therefore \overbrace{(w_0 - w_1)}^{m}\, x = \frac{\overbrace{(wk_1 - wk_0)}^{c}}{2}$$

So, the values of slope & intercept can be given as,

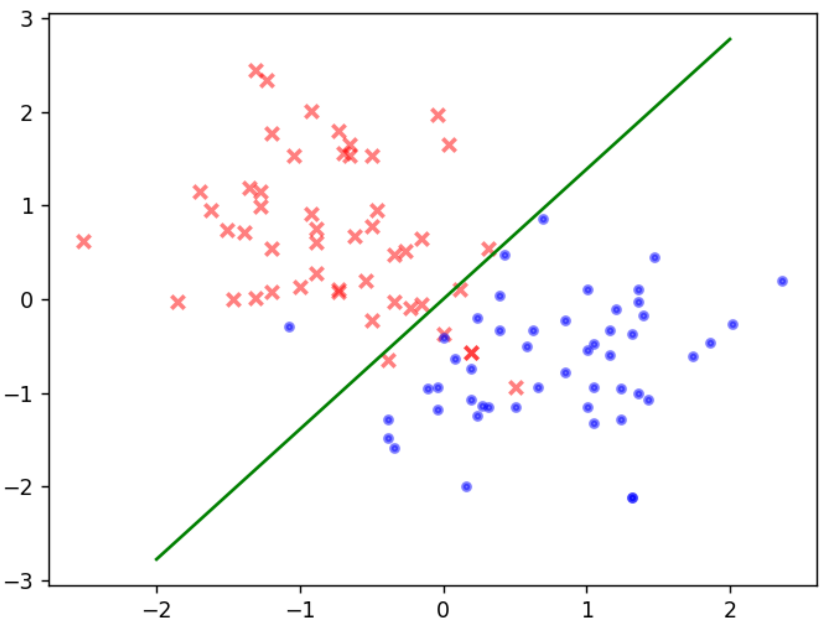
$$m = -\frac{(w_0(0) - w_1(0))}{w_0(1) - w_1(1)}$$

$$c = \frac{1}{2}\,\frac{(wk(0) - wk_1)}{w_0(1) - w_1(1)}$$

As the quantities w_0 and w_1 are matrices and are indexed, we need to use indexed values' to

(b) Then, plotted the decision boundary.

(c) The Decision Boundary
The Red "X" markers represent Alaska and Blue "." Markers represent Canada.



(d) The values of parameters obtained

μ_0 = [0.755294 -0.685094]

μ_1 = [-0.755294 0.685094]

Σ_0 = [0.477471 0.109920]

[0.109920 0.413554]

Σ_1 = [0.381589 -0.154865]

[-0.154865 0.647737]