Report for Assignment 1 of COL 774

Ayushi Agarwal
2018ANZ8503
ayushi.agarwal@cse.iitd.ac.in

1 Introduction

The goal of this assignment is to implement four popular Machine Learning algorithms which include:

- Linear Regression
- Stochastic Gradient Descent or SGD
- Logistic Regression
- Gaussian Discriminant Analysis, henceforth referred as GDA.

In the next section of this report, we will discuss about the algorithm in each question, our method to implement the algorithm, report our results and present an analysis of the results obtained.

2 Method, Results and Analysis

In this section, we would discuss our algorithm implementation and the results obtained. We would present an analysis of the results as well.

2.1 Question 1 - Linear Regression

Linear Regression is one of the most popular algorithms used in Machine Learning used to learn linear models on data. In this question, we implement a variant of linear regression, i.e. the least squares linear regression to predict the density of wine based on its acidity. The feature $x \in \mathbb{R}$ is "Acidity of the wine" and the output $y \in \mathbb{R}$ is "Wine Density". We have normalized the data to be of zero mean and unit variance so that the features are not of very different scales. It helps the model to converge faster.

We have used batch gradient descent technique to optimize the least squared error function or the cost function $J(\theta)$ in this question which can be defined as follows:

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

We have implemented the average cost otherwise the algorithm requires a very small learning rate to converge.

The initial value of the model parameter θ is zero.
The stopping criteria of the algorithm is based on the value of the gradient of the cost function which is $\nabla_\theta J(\theta)$. We stop our algorithm as soon as the value of the gradient becomes very close to zero.

$$\nabla_\theta J(\theta) = X^T.(h_\theta(X) - Y) \tag{2}$$

where

$$h_\theta(x) = X.\theta \tag{3}$$

The termination condition used is :

$$\left|\nabla_\theta J(\theta)\right| <= 1e^{-7}$$

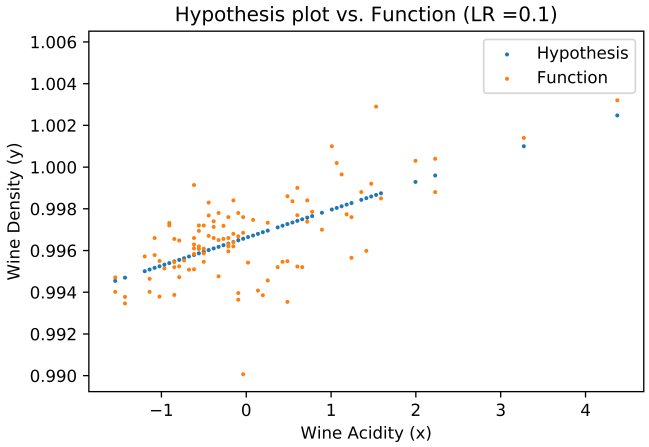
This condition is enough for our model even though it guarantees only local minima. But the objective function that we are optimizing is a convex function so this condition is enough for our data.
The learning rate is a very important hyper-parameter of any machine learning model because it decides the step size of the parameter update. A very small learning rate can lead to very slow convergence of the model i.e minimizing or maximizing the objective function. On the other hand, a very high learning rate can lead to divergence of the cost function. So, we have experimented with the learning rate to find the appropriate learning rate and analyse at what learning rate the model diverges.

The most optimized learning rate for the data is $LR = 0.1$. The model also converges on $LR = 0.01$ but the learning is slower and the model takes time to converge to the minimum as compared to $LR = 0.1$.The learning becomes faster as we increase the learning rate but as we increase the learning rate to $LR >= 1.1$, the parameters start to oscillate between opposite gradient before converging. The model starts diverging at $LR > 1.9$. The hypothesis is almost the same for all the learning rates except when $LR > 1.9$.

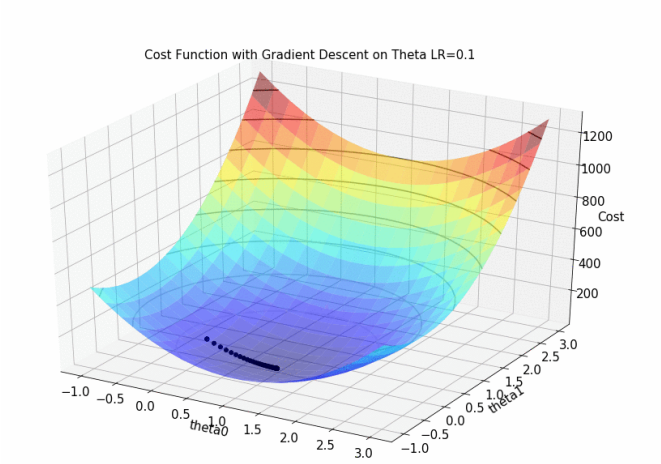
$$\theta_{final} = [0.99661337, 0.00134019]$$

We report our results only for $LR = 0.1$[I]. Figure 1

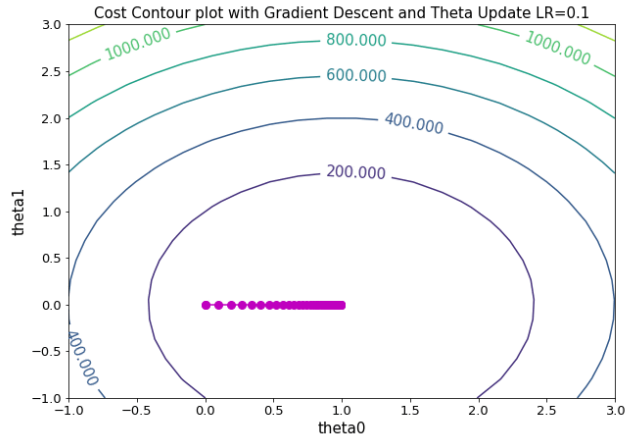
[I] https://github.com/agarwal-ayushi/Machine-Learning-Assignments/tree/master/Assignment1



(a) Data and $h_\theta(x)$ for Linear Regression



(b) Cost Function plotted w.r.t to θ_0 and θ_1

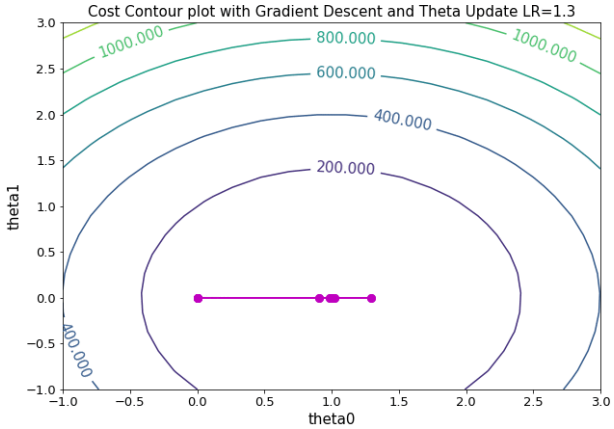


(c) Contours of the cost function at LR=0.1

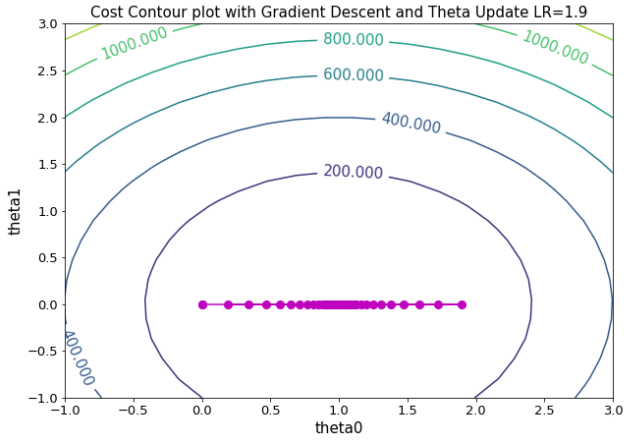
Figure 1: Least Square Linear Regression at Learning Rate=0.1

shows the results that we obtained for running gradient descent for optimizing least square linear regression. Figure 1a shows the hypothesis learnt on the provided data at $LR = 0.1$. Figure 1b shows the 3-D plot of the cost function with respect to the learned parameters plotted at every step of the gradient descent to show the descent in the cost. Figure 1c shows the corresponding contour plot with the parameter updates at every step of gradient descent.

Figure 2a and 2b show the contour plot for higher learning rate $LR = 1.3$ and $LR = 1.9$. It shows the parameters oscillating around the minima before finally converging. At higher learning rate's the model diverges.



(a) Contours of the cost function at LR=1.3



(b) Contours of the cost function at LR=1.9

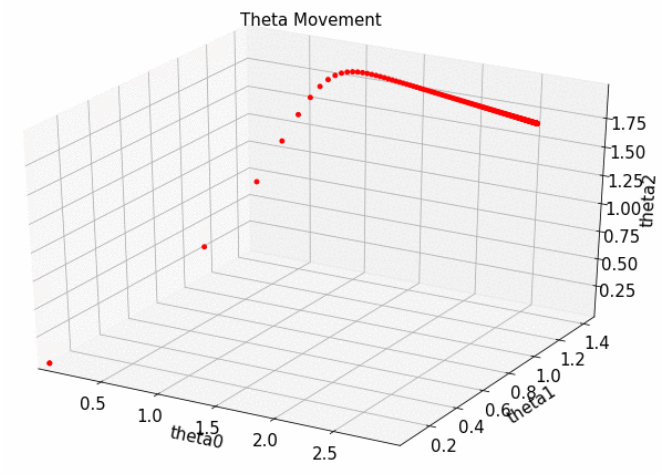
Figure 2: Contour plots for Least Square Linear Regression at Learning Rate=1.3 and 1.9

2.2 Question 2: Stochastic Gradient Descent

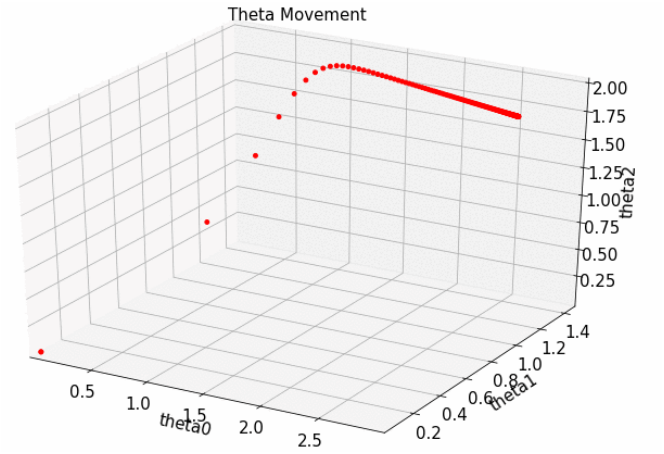
In this question, we will use probabilistic interpretation of linear regression to generate the data and then apply stochastic gradient descent to optimize the cost. We assume that the hypothesis is given to us as in equation 3. We sample the $y \in \mathbb{R}$ by adding Gaussian noise to the available hypothesis. This will generate the data $(x^{(i)}, y^{(i)}) \forall i \in (1, m)$. The Gaussian Noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
The $y^{(i)}$ would be sampled as:

$$y^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \epsilon^{(i)}$$

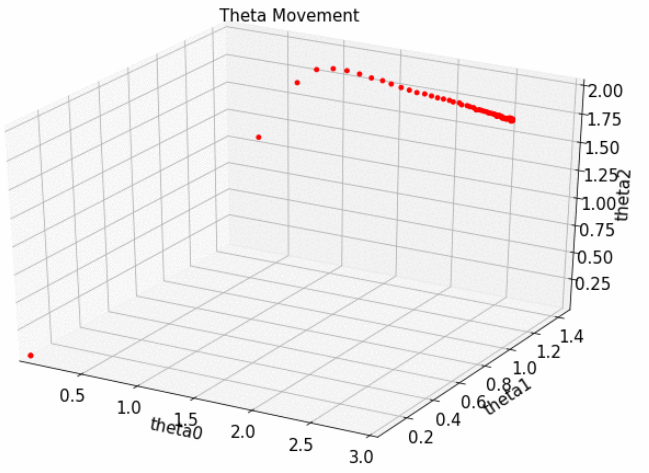
We will then use SGD to learn the original hypothesis from the data that was generated from the above sampling. We have implemented the version where we make a complete pass through the data in a round-robin fashion to choose r samples at a time. We also report results on how the convergence changes by varying the batch size r. The cost function is the same as in equation 1, only we don't go through all m samples but rather just go through



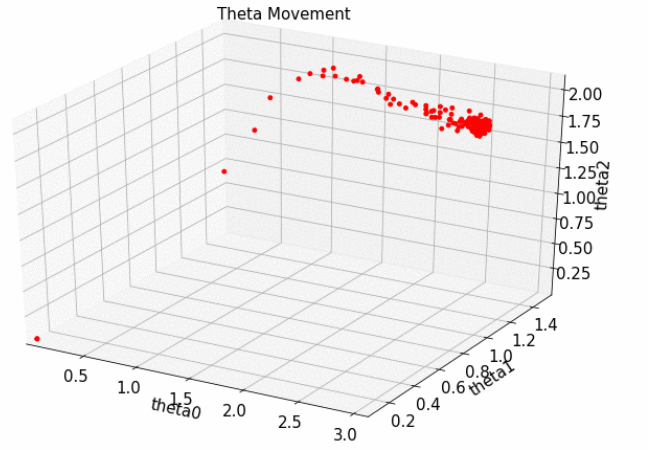
(a) Theta update when batch size = 1000000



(b) Theta update when batch size = 10000



(c) Theta update when batch size = 100



(d) Theta update when batch size = 1

Figure 3: Stochastic Gradient Descent with different batch sizes on 1 million points

r samples to update the parameters. This leads to faster convergence.

Part a:

We have sampled 1 million data points using $\theta = [\theta_0, \theta_1, \theta_2] = [3, 1, 2]$, $x_1 \sim \mathcal{N}(3, 4)$, $x_2 \sim \mathcal{N}(-1, 4)$ and the noise variance $\sigma^2 = 2$

Part b:

We use stochastic gradient descent with different batch sizes to learn the original hypothesis. The learning rate $LR = 0.001$ is fixed for all the batch sizes $r \in \{1, 100, 10000, 10000\}$ because of the objective to compare their performance. The convergence criteria for different batch sizes and the theta learnt in each case is:

- **Batch Size = 1000000**
  In this case, the batch contains the entire dataset so this is equivalent to batch gradient descent where there is only one update per epoch. The update step is based on the cost of the entire sample. **Hence, this model converges slowly but less randomly**.
  The smooth path of the theta parameter convergence can be seen in Figure 3a. *(The path can be seen in motion in the GIF file that we have submitted). The convergence criteria is : *we stop when the change in the cost for the entire dataset before and after the epoch is smaller than* $1e-7$. It takes around **20k epochs** to converge at this stopping criteria.
  $\theta_{final} = [2.98784635, 1.00189187, 1.99937068]$
  This is same as the initial parameters that we used to sample the data.
  The Test error on Original Hypothesis with theta 0 = 0.9829469215
  The Test error on Learned Hypothesis = 0.9832330380748425
  The difference in the test error of original and learned hypothesis is = 0.00028611657484256536
  The test error is very small in this case as it is similar to least squared linear regression.

- **Batch Size = 10000**
  Figure 3b shows the path of the theta convergence for a batch size of 10000 out of a 1 million sample space. The path is similar to batch gradient descent but the **convergence is a little faster than batch size of 1M**. The convergence criteria is : *we stop when the change in the cost for the entire dataset before and after the epoch is smaller than* $1e-7$. It takes around **289 epochs** to converge to the same parameters.

$\theta_{final} = [2.99811017, 0.99970481, 2.00024497]$
This is the same as the original parameters as well. But it is relatively faster.
The Test error on Original Hypothesis with theta 0 = 0.9829469215
The Test error on Learned Hypothesis = 0.9829537652379439
The difference in the test error of original and learned hypothesis is = 6.8437379439068025e-06
The test error is very small as the batch is still very large and little randomness is there.

- **Batch Size = 100**
  Figure 3c shows the path of gradient descent of the parameter for a batch size of 100 samples. So the update is based on 100 samples at a time. **However, there is one observation that the path of the convergence is not as smooth as the other larger batch sizes.** This is because updates based on small batches are **more random** and the cost is a stochastic approximation of the actual cost function over the entire dataset.The update is much faster than the larger batch sizes so the model converges in around 6 epochs with the same convergence criteria.
  $\theta_{final} = [2.99799309, 0.99888929, 2.0025575]$
  This is the same as the original parameters but the **parameter convergence is very fast**. The Test error on Original Hypothesis with theta 0 = 0.9829469215
  The Test error on Learned Hypothesis = 0.9830876293940337
  The difference in the test error of original and learned hypothesis is = 0.0001407078940337536
  The test error is also small since we don't see very high randomness.

- **Batch Size = 1**
  When the batch size is one, the parameters are updated looking at the gradient of the cost function for each data sample. So the algorithm takes 1 million small steps per epoch. Figure 3d shows the convergence path of the θ parameter. The **convergence is extremely fast** since we don't go through the entire dataset to do one update. The model converges within **1 epoch**.
  However, we also notice that the **path is highly random** as compared to the larger batch sizes because of the stochastic randomness since we are just taking the cost at one data sample to do the update instead of the total cost.
  $\theta_{final} = [3.00555871, 1.02308559, 1.94557533]$
  The Test error on Original Hypothesis with theta 0 = 0.9829469215
  The Test error on Learned Hypothesis = 1.1696159019660004
  The difference in the test error of original and learned hypothesis is = 0.18666898046600044.
  The **test error is a bit higher** in this case because of the randomness in the updates. The model doesn't actually converge to the exact minima but keeps on meandering around it. To make the model converge, the learning rate should be decreased as the iterations increase.

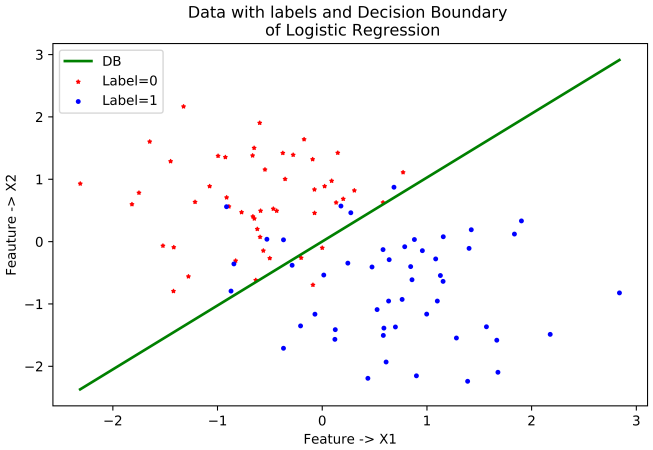


Figure 4: Decision boundary learnt by Newton's Method for Logistic Regression

2.3 Question 3 - Logistic Regression

Logistic Regression is a classification model used to classify different classes of data. In this section, we use the Newton's Method for optimizing the negative log-likelihood of the cost function of logistic regression. The given data has two class labels - 0 and 1. There are two input features i.e. $x \in \mathbb{R}^2$ and $y \in \{0, 1\}$. We have normalized both the features to be of zero mean and unit variance to make them lie in the same scale. The log-likelihood of the logistic regression is:

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

where

$$h_\theta(x) = \sigma(X.\theta) \tag{5}$$

Equation 2 shows the gradient of the cost function for linear regression. Since logistic and linear regression both belong to the family of generalized linear models, the cost function takes the same form but the hypothesis functions are different as seen in equations 3 and 5.

Newton's Method uses the second order derivative for optimizing the log-likelihood so we calculate the Hessian of the gradient of the log-likelihood of logistic regression as shown in equation 2.

$$H_\theta(LL(\theta)) = X^T.diag(\sigma(X.\theta)(1 - \sigma(X.\theta))).X \tag{6}$$

We initialize the Newton's method with $\theta = [0, 0, 0]$. After calculating the hessian, we update θ according to the

Newton's update step :

$$\theta^{t+1} = \theta^t - H^{-1}\nabla_\theta(LL(\theta)) \tag{7}$$

Figure 4 shows the decision boundary learnt by logistic regression using Newton's Method to optimize the log-likelihood of the cost function. The final parameter obtained from this optimizing step is: $\theta_{final} = [-0.00064394, 0.00921424, -0.00898329]$.

2.4 Question 4: Gaussian Discriminant Analysis

In this section, we will present the results obtained by implementing GDA for separating out class from salmons from Alaska and Canada. We have two features to represent each salmon so $x \in \mathbb{R}^2$ and $y \in \{'Alaska','Canada'\}$. We have normalized both the features to be of zero mean and unit variance to make them lie in the same scale. We implement GDA using the closed form equations obtained by equating the log-likelihood of generating the entire dataset in GDA to zero. Since it is a binary classifier, we learn two different normal distributions for the two classes with different means and covariance. If the covariance for both the classes is same, then we get a linear decision boundary, otherwise we get a quadratic decision boundary. We have allotted the label 'Alaska' to class 1 and label 'Canada' to class 0. The equations used are:

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

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

$$\sum = \frac{\sum_{i=1}^{m}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T}{m}$$

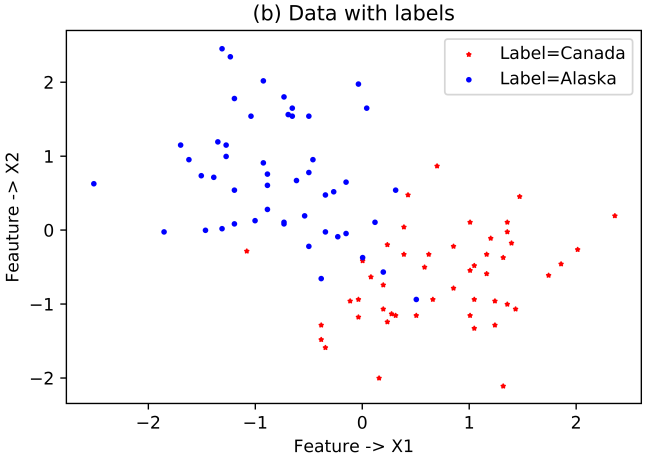
Part a: We assume that both the classes have the same covariance. So we get a linear separator.
The values of the mean and covariance are:
$\mu_0 = [0.75529433, -0.68509431]$,
$\mu_1 = [-0.75529433, 0.68509431]$ and
$\sum = \begin{pmatrix} 0.42953048 & -0.02247228 \\ -0.02247228 & 0.53064579 \end{pmatrix}$

Part b: Figure 5a shows the training data corresponding to the two coordinates of the input features and the corresponding classes that they belong to.

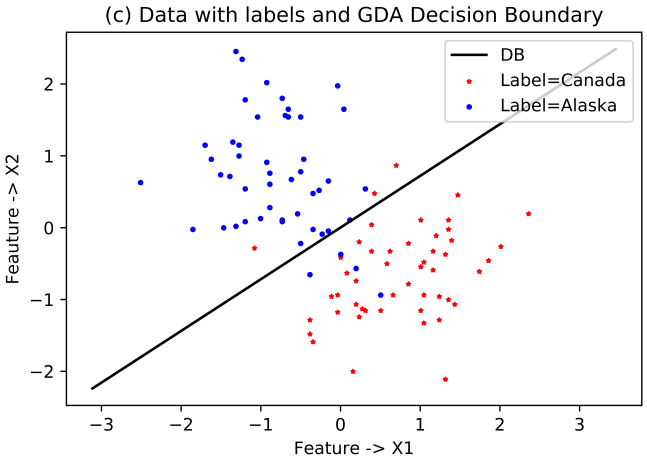
Part c: Figure 5b shows the linear boundary learnt by the GDA model when the covariance of both the classes are identical.

Part d: When both the classes have different covariance matrices, then the model learns a quadratic boundary. The values of the covariance learned by our implemented model is:
$\sum_0 = \begin{pmatrix} 0.47747117 & 0.1099206 \\ 0.1099206 & 0.41355441 \end{pmatrix}$

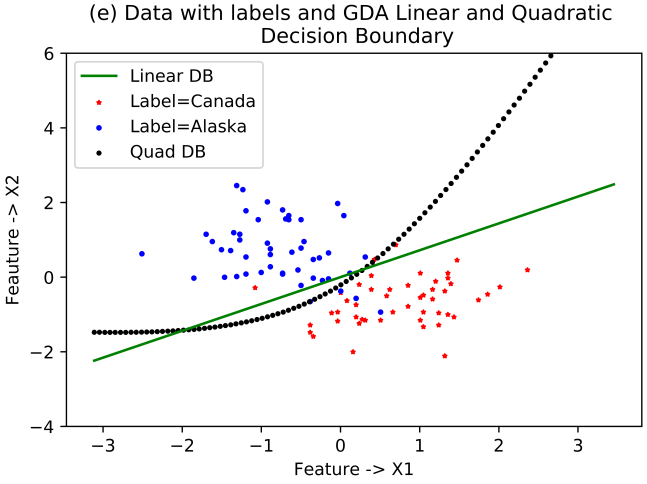
$\sum_1 = \begin{pmatrix} 0.38158978 & -0.15486516 \\ -0.15486516 & 0.64773717 \end{pmatrix}$



(a) Training Data with two labels



(b) Linear Decision Boundary Learnt by GDA when $\sum_0 = \sum_1$



(c) Quadratic Decision Boundary Learnt by GDA when x_1 and x_2 have different $\sum$ matrices

Figure 5: Decision boundary learnt by Newton's Method for Logistic Regression

The values of the mean of the two distributions remain the same.

Part e: Figure 5c shows the quadratic boundary learnt by the GDA model by using the parameters $\mu_0, \mu_1, \sum_0 and \sum_1$.

Part f: The linear boundary learnt by the GDA is not a very optimal separator of the two classes. It can clas-

sify the data with high accuracy but the data is not well-separated by this linear boundary. So the probability of a new point to be classified to belong to a certain class will not be very high which means we would not be able to classify points with high certainty. This is because of the lack of covariance information between the two classes. This makes it equivalent to logistic regression.
On the other hand, the quadratic decision boundary separates the data well because it has the knowledge of how the two classes are correlated and so it is a good separator for this data.

3 Conclusions

This report is a summary of the first assignment done for Machine Learning course. We obtain a good understanding of how the algorithms that we talked about in this report are implemented and we also analyzed the affect of different hyper-parameters on the learning.