COL774 Assignment-1

Shrey J. Patel, 2019CS10400

September 2021

1 Linear Regression

(a) Final Values:

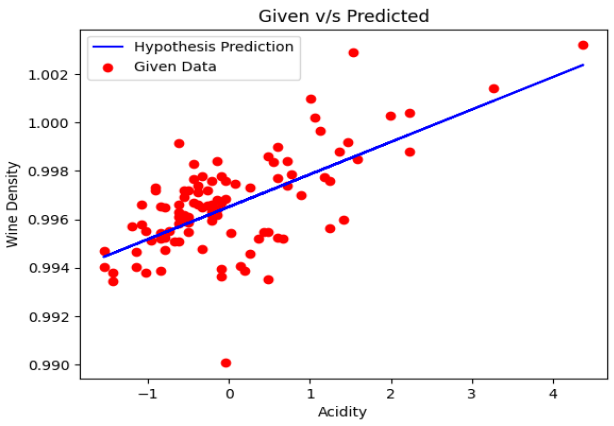
- **Learning Rate:** 0.001
- **θ:** [0.99652102 0.00134006]
- **Number of iterations:** 8053
- **Stopping Criterion:**
  The desirability of the parameters is decided on the basis of the the corresponding cost function J(θ), which is modelled as the Least Square function as follows:

  $$J(\theta) = \left(\frac{1}{2m}\right) \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$$

  Although we desire to reach zero error, it is not practical since we want to learn the data and not memorise it, so we stop not at some particular value of J(θ), but when the rate of decrease in cost function becomes insignificant, which we parameterize using ϵ. We have taken $\epsilon = 10^{-10}$. So, our convergence criteria is:

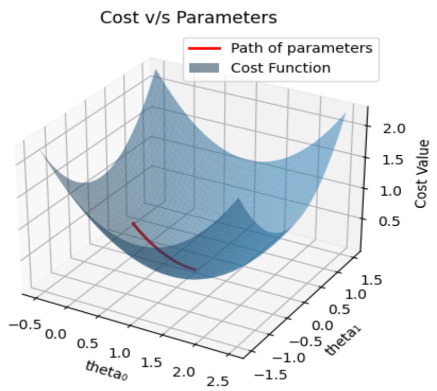
  $$J(\theta^t) - J(\theta^{t+1}) \leq \epsilon$$

(b) The plot for comparing the given data and the trained hypothesis:



As evident from the graph, the hypothesis(blue) tries to pass approximately from the middle of all the data points so as to obtain the best fit i.e. the minimum error.

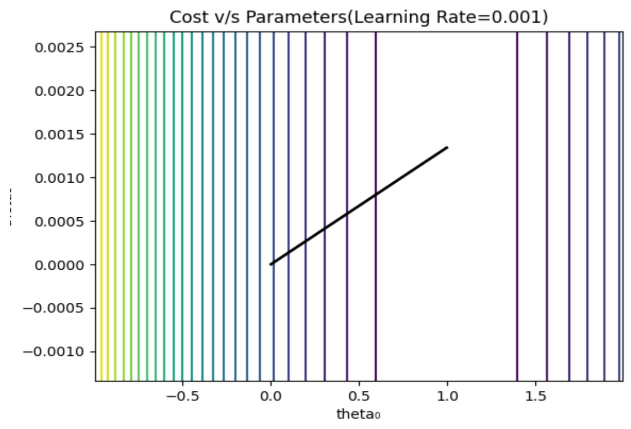
(c) Mesh plot for comparing the path that the parameters take during the learning algorithm against the cost function/error function:



From the graph, it can be seen that the path on the cost surface that the parameters take to reach convergence(which is near the local minimum, since the cost function changes the least there), is against the direction of the steepest gradient, so as to reach the convergence faster. Thus, this is a fairly visual and intuitive representation of Gradient Descent algorithm.

Additionally, changing the values of learning rate do not change the actual path, only the number of iterations to reach the convergence change, which can be observed in the speed of the animation(not shown here)

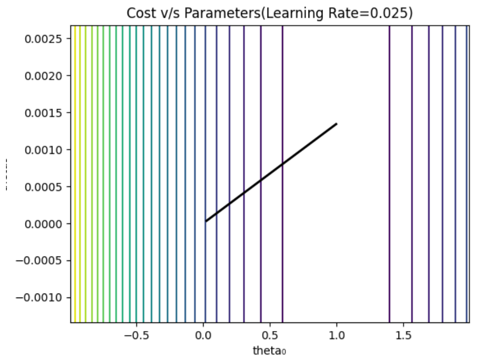
(d) Contour plot for comparing the path that the parameters take during the learning algorithm against the cost function/error function:



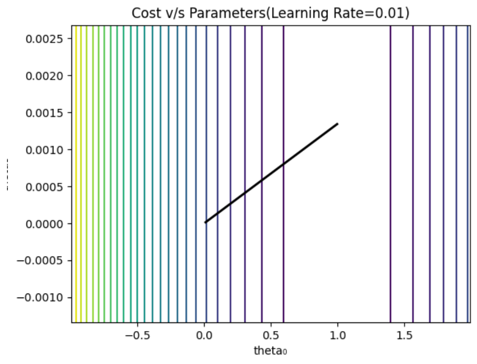
The contour plot provide a top view(i.e. a 2D view) of the path of parameters. Unlike the mesh plot, this can be used to obtain the final value of the paramaters, which from the graph are roughly [1 0.0015], which is close to the obtained value.

The lines in the contour plot signify the values which are at the same height in the mesh plot. The increasing distance between may be representative of the fact that the cost function i.e. J(θ) is a convex function, since the gradient keeps on decreasing near the local minima. So, the same decrease in height(which is denoted by the lines) can be achieved by a greater change in the parameters, and thus the increasing distance between the lines.

(e) Contour plots for different learning rates:



(a) η = 0.025



(b) η = 0.01

Notice that, as argued before in part (c), the path taken for all three learning rates are same, along with the final parameter values:

- **η = 0.025:**
  θ = [0.99655882 0.00134011]
  Number of iterations = 383
- **η = 0.01:**
  θ = [0.99652102 0.00134006]
  Number of iterations = 917

On comparing this, we can observe that the final parameter values and therefore the final cost value are same regardless of the learning rate. The only difference is in the number of iterations taken to converge. The general trend which can be observed is that higher the learning rate, faster it is to reach convergence(and so lower the number of iterations) provided that we don't overshoot the minima.

2 Sampling and Stochastic Gradient Descent(SGD)

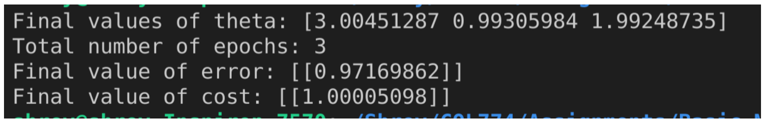
(a) The given feature values i.e. $x_1 = \mathcal{N}(3,4)$ and $x_2 = \mathcal{N}(-1,4)$ can be easily sampled using the in-built numpy function i.e. *np.random.normal*. However, since the theta vector i.e. $\theta = \theta_0\ \theta_1\ \theta_2 = (3\ 1\ 2)$ is given, we can use the hypothesis equation to generate y i.e.

$$y = h_\theta(x) = \theta_0 + \theta_1 * x_1 + \theta_2 * x_2 + \epsilon$$
or
$$y = \theta^T X + \epsilon \text{ where X} = (x_0 = 1\ x_1\ x_2).$$

where $\epsilon = \mathcal{N}(0,1)$

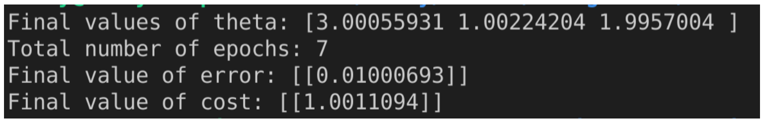
(b) The final values of the parameters for each of the batch sizes are :

- **Batch size = 1**:



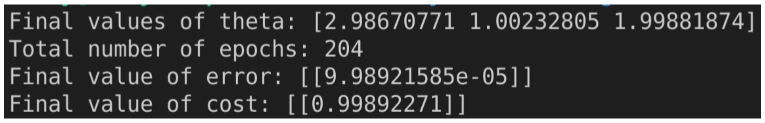
```
Final values of theta: [3.00451287 0.99305984 1.99248735]
Total number of epochs: 3
Final value of error: [[0.97169862]]
Final value of cost: [[1.00005098]]
```

- **Batch size = 100**:



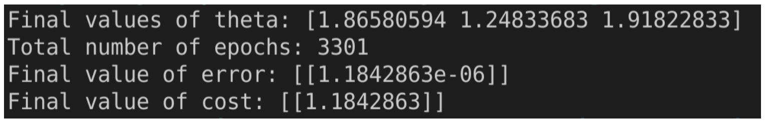
```
Final values of theta: [3.00055931 1.00224204 1.9957004 ]
Total number of epochs: 7
Final value of error: [[0.01000693]]
Final value of cost: [[1.0011094]]
```

- **Batch size = 10000**:



```
Final values of theta: [2.98670771 1.00232805 1.99881874]
Total number of epochs: 204
Final value of error: [[9.98921585e-05]]
Final value of cost: [[0.99892271]]
```

- **Batch size = 1000000**:



```
Final values of theta: [1.86580594 1.24833683 1.91822833]
Total number of epochs: 3301
Final value of error: [[1.1842863e-06]]
Final value of cost: [[1.1842863]]
```

Convergence Criteria:
Here, we use a convergence criteria similar to the the first question, i.e. we check if the difference in the cost functions of consecutive iterations is less than some ϵ. However, since we don't use the entire data for updating the parameters at a time, we also don't consider all the data points in computing the cost function. The updated cost function is(given batch size=b):

$$J_r(\theta) = \left(\frac{1}{2b}\right) \sum_{k=b*r}^{b*r+b-1} (y^{(i_k)} - \theta^T x^{(i_k)})^2$$

So, in each iteration, we calculate the cost function for the current batch. But since it is always better to consider the entire data for convergence, we calculate these cost functions over all batches and take their average to obtain the cost value of an epoch, which is:

$$J_e(\theta) = \left(\frac{1}{m}\right) \sum_r J_r(\theta)$$

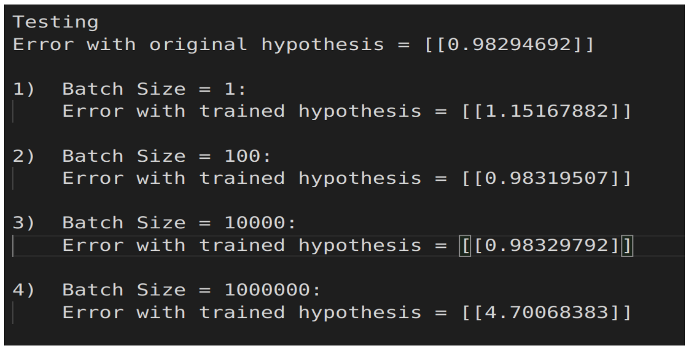
And it is this $J_e(\theta)$ that we will use to decide convergence. So, our final convergence criteria is:

$$J_e(\theta^t) - J_e(\theta^{t+1}) \leq \epsilon$$

(c) **Observations:**

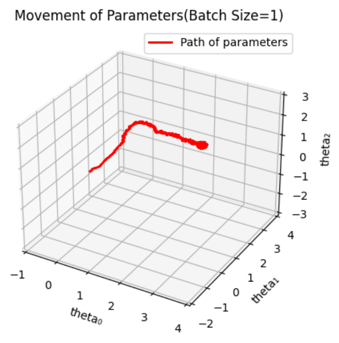
- The first three batch sizes (except 1M) converge to the same parameter values as observed above, and these values are almost same as the given hypothesis value. However, the parameter values in the 1M batch size case are wildly off from the given hypothesis.
- Note that higher the batch size, lesser will be the number of batches in the data set of fixed size, and so lesser will be the number of iterations per epoch. So, if we assume for a minute that for a given learning rate(i.e. $\eta = 0.001$), all batch sizes take equal number of iterations to converge, then higher batch sizes will take higher number of epochs.
- For batch size 1, each iteration uses only one data point for learning in each iteration, due to which there are many fluctuations in the path of the parameters, as observable from the graph below. So, the algorithm takes the greatest time(in terms of iterations) to reach convergence, since it may take more iterations for the average cost of an epoch to smoothen out. But, on the other hand, the time taken for each iteration is also less.
- On the other hand, for batch size 1M, an iteration is same as an epoch and so each iteration is very expensive. Also, note that this case is the only case in which the final parameter value is very off from the original hypothesis and consequently, the test data will have high error value. It can also be observed from the graph that the algorithm converges in much less iterations than the other cases. The main reason behind this is because this case is exactly similar to the classic gradient descent, and so the parameters always move in the fastest way possible i.e. against the gradient of global cost function.
- However, for batch sizes 100 and 10K, both these factors, i.e. the complexity of each iteration and the number of iterations are balanced out, so as to yield a better running time.

The summary of testing is:

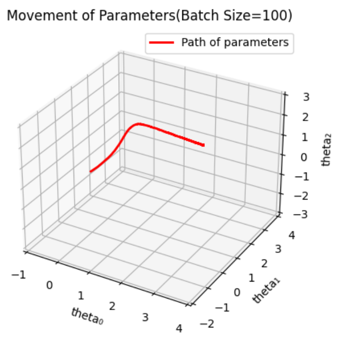


```
Testing
Error with original hypothesis = [[0.98294692]]

1)  Batch Size = 1:
    Error with trained hypothesis = [[1.15167882]]

2)  Batch Size = 100:
    Error with trained hypothesis = [[0.98319507]]

3)  Batch Size = 10000:
    Error with trained hypothesis = [[0.98329792]]

4)  Batch Size = 1000000:
    Error with trained hypothesis = [[4.70068383]]
```

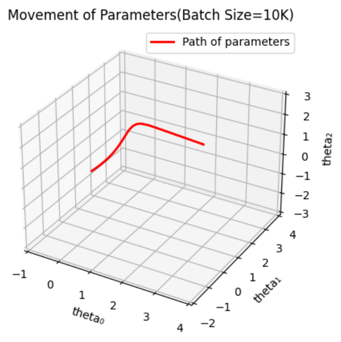
(d) The plots for parameter paths for different batch sizes are:



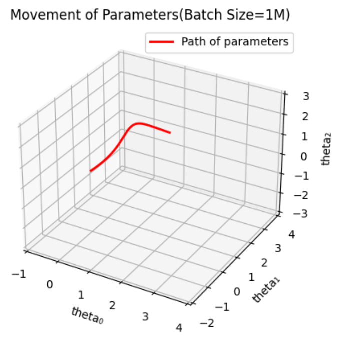
(a) Batch Size = 1



(b) Batch Size = 100



(a) Batch Size = 10K



(b) Batch Size = 1M

3 Logistic Regression

(a) For the given log-likelihood function:

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

we try to find $argmax_\theta LL(\theta)$ using Newton's update method i.e.

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

The gradient can be obtained by differentiating LL($theta$) as:

$$\nabla_\theta LL(\theta) = \sum_{i=1}^{m} (y^{(i)} - \left(\frac{1}{1+e^{-\theta^T x}}\right)) * x^{(i)}$$

The Hessian matrix on the other hand can be found from second order partial derivatives as follows:

$$\mathcal{H}_{ij} = \left(\frac{\partial^2 LL(\theta)}{\partial\theta_i \partial\theta_j}\right) = \sum_{k=1}^{m} \left(\frac{-e^{-\theta^T x^{(k)}}}{(1+e^{-\theta^T x^{(k)}})^2}\right) * (x_i)^{(k)} * (x_j)^{(k)}$$

or alternatively

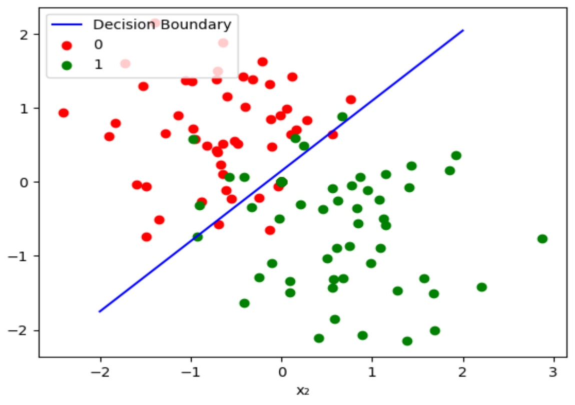
$$\mathcal{H} = X^T D X$$

where X = Feature Vector and $D = diag(f(1), f(2), ...f(m))$, where

$$\text{f(k)} = \left(\frac{-e^{-\theta^T x^{(k)}}}{(1+e^{-\theta^T x^{(k)}})^2}\right)$$

Using the above update method, the final parameter values obtained after training are:
θ: [0.40125316, 2.5885477, -2.72558849]
Number of iterations: 9

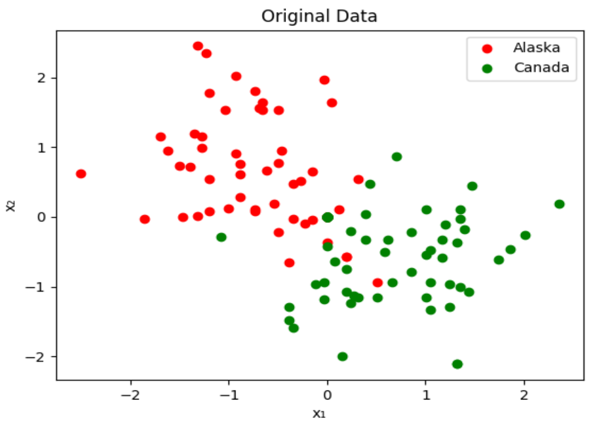
(b) Plot for given data and learned separator using Newton's Method:



4 Gaussian Discriminant Analysis(GDA)

(a)
- ϕ**:** 0.5
- μ_0**:** [-0.75529433 0.68509431]
- μ_1**:** [0.75529433 -0.68509431]
- Σ**:** [[0.42953048, -0.02247228] [-0.02247228, 0.53064579]]

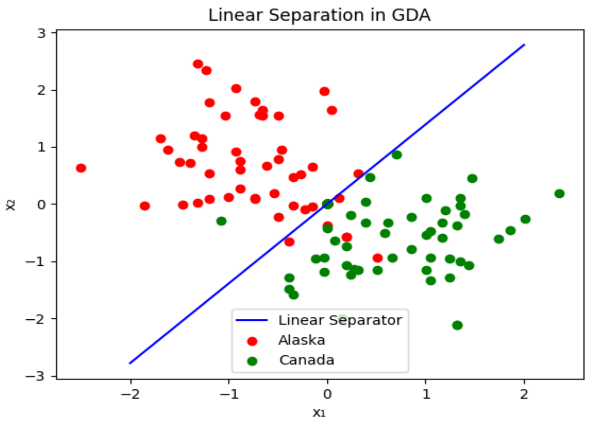
(b) Plot of input data with countries as labels:



(c) Equation of linear separator:

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

The following plot shows the linear separator against the input data to measure the effectiveness of separation:

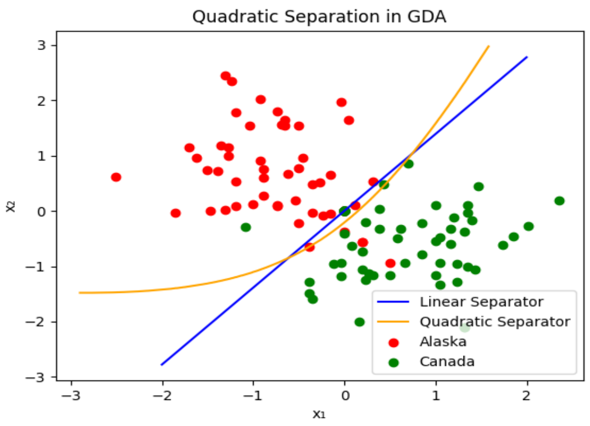


(d)
- μ_0**:** [-0.75529433 0.68509431]
- μ_1**:** [0.75529433 -0.68509431]
- Σ_0**:** [[0.38158978, -0.15486516] [-0.15486516, 0.64773717]]
- Σ_1**:** [[0.47747117, 0.1099206] [0.1099206, 0.41355441]]

(e) Equation of quadratic separator:

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

The following plot shows the quadratic separator and the linear separator against the data points to compare the effectiveness of separation for both the curves.



(f) **Comparison between Linear and Quadratic Separators:**

- Both the separators separate the given data set very well with naturally some wrong predictions as we need to learn the data and not memorise it.
- However, if we observe the region in the middle, then some salmons from Alaska(red) which were wrongly classified into Canada by the linear separator, are now correctly classified into Alaska by the quadratic separator because of the ”bend” across these contested points.
- Although we may not be sure if the quadratic separator overfits due to the less number of data available, since not all the points from the data set are classified accurately by the quadratic separator, but we may be sure that the quadratic separator favours Canada over Alaska since it splits the space into regions of unequal areas, possibly just to favour a few outliers.
- While the linear separator doesn’t bias any region while still fitting the data satisfactorily. So, linear separator may be better.