ECE521

ASSIGNMENT 2:

LINEAR AND LOGISTIC REGRESSION

REPORT

AUTHORS:

Smeet Gajjar –

Nithin Ganesh Prasad – 999 676 426

Tony Ngo -

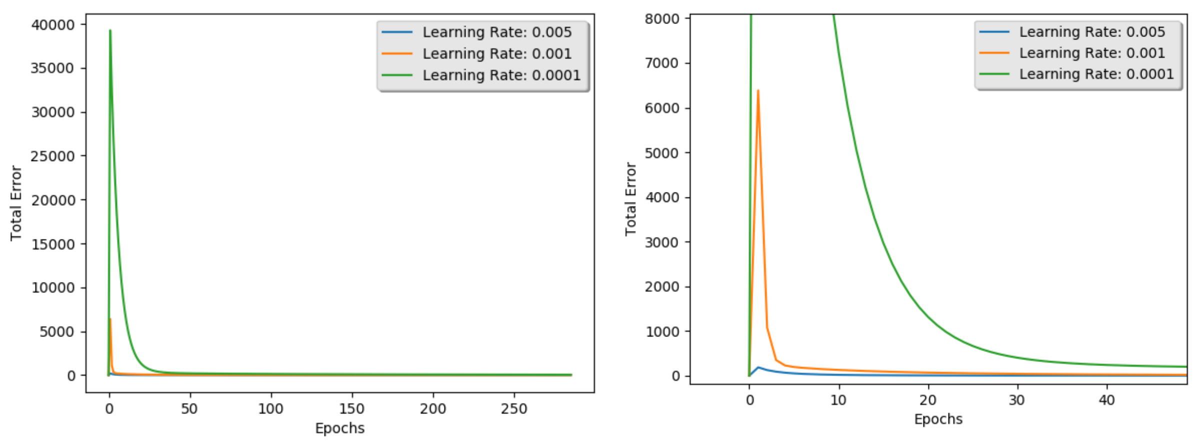
PART 1: LINEAR REGRESSION

General Comments:

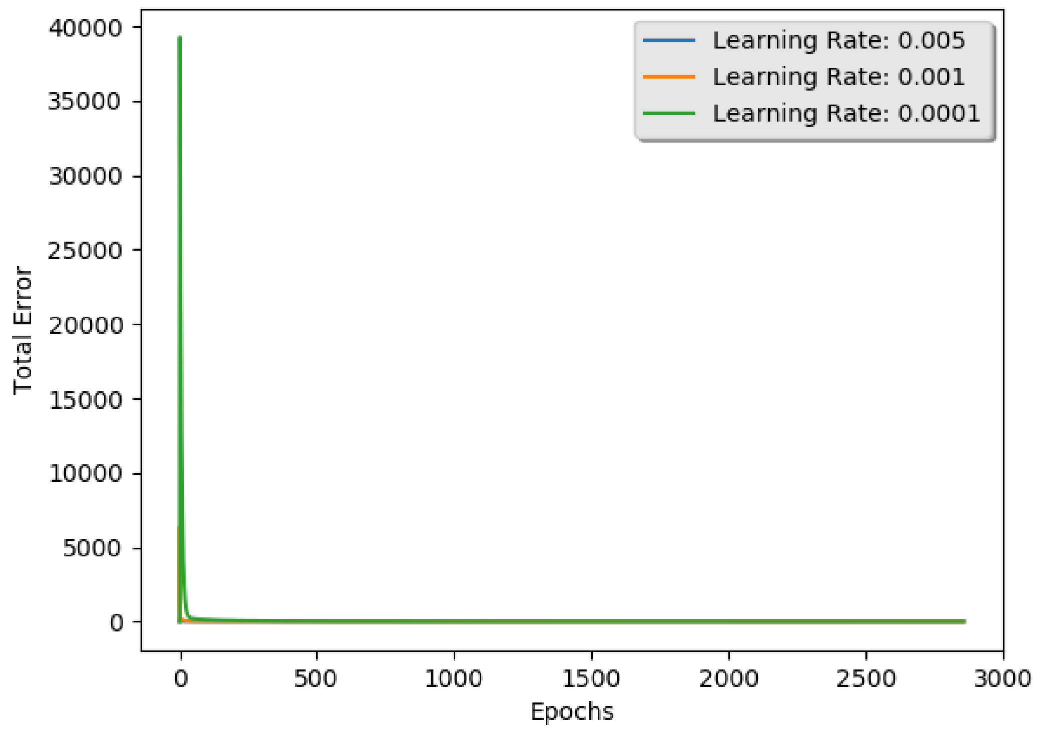
Upon consulting resources online, a variant of Stochastic Gradient Descent was found, in which the entire dataset was randomized in every epoch. This was seen as an interesting suggestion as it means that the gradient computed (for instance on 7 batches over 3500 training data instances) will be done on a different data set each time; thus randomization at this step would lead to increased generalization. In our preliminary algorithm, we did not apply this variant, but we will include this in our final submission, time-permitting.

1.1 Tuning the Learning Rate:

Plot of Loss function vs. Number of Epochs for Learning Rates:
η = 0.005, η = 0.001, η = 0.0001
for Linear Regression with SGD (with Mini-Batch)



- Both of the above figures depict the various learning rates for Linear Regression taken over 2000 iterations; the figure on the right is a zoomed version of the figure on the left.
- As is evident by the right figure, an increase in learning rate results in faster convergence of the error on the training data. In this case, *the optimal learning rate is 0.005*.



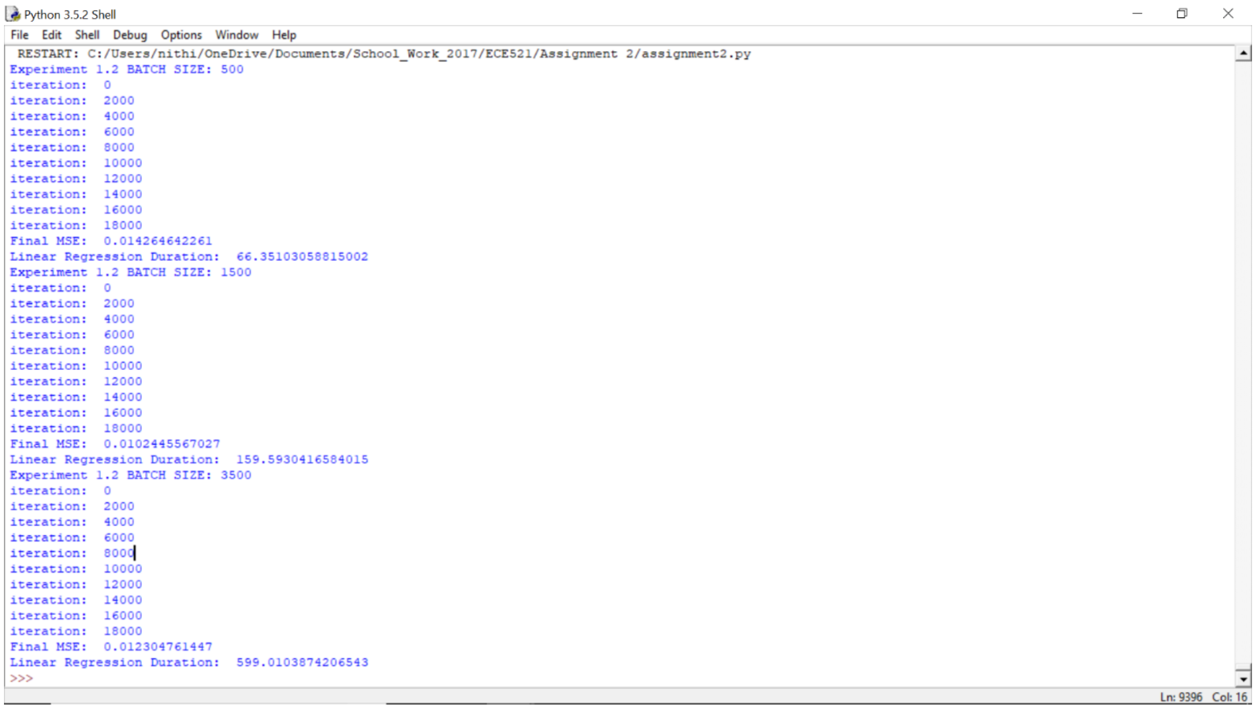
- This graph depicts the same data but for 20000 iterations (as specified in the assignment). However, this graph is only included for completion, as the details here are obscure relative to the clarity provided by the two preceding graphs.

<u>1.2 Effect of the Mini-Batch Size:</u>

- From above, the learning rate was chose to be 0.005.
- From the image (screen-shot) attached, we note the following:

BATCH SIZE	**500**	**1500**	**3500**
Execution Time [seconds]	66.35	159.59	599.01
Total Error (on Iteration 19999)	0.01426	0.01024	0.01230

- The *best mini-batch size in terms of purely training time is observed to be batch size: 500*. However, we notice that a slight increase in training-data can be obtained by choosing a batch size of 1500, at the cost of a small multiplicative factor of increase in the computation cost for 20000 iterations. The data makes sense because we expect the gradient to be more computationally heavy when trying to optimize more data-points simultaneously.



```
 RESTART: C:/Users/nithi/OneDrive/Documents/School_Work_2017/ECE521/Assignment 2/assignment2.py
Experiment 1.2 BATCH SIZE: 500
iteration:  0
iteration:  2000
iteration:  4000
iteration:  6000
iteration:  8000
iteration:  10000
iteration:  12000
iteration:  14000
iteration:  16000
iteration:  18000
Final MSE:  0.014264642261
Linear Regression Duration:  66.35103058815002
Experiment 1.2 BATCH SIZE: 1500
iteration:  0
iteration:  2000
iteration:  4000
iteration:  6000
iteration:  8000
iteration:  10000
iteration:  12000
iteration:  14000
iteration:  16000
iteration:  18000
Final MSE:  0.0102445567027
Linear Regression Duration:  159.5930416584015
Experiment 1.2 BATCH SIZE: 3500
iteration:  0
iteration:  2000
iteration:  4000
iteration:  6000
iteration:  8000
iteration:  10000
iteration:  12000
iteration:  14000
iteration:  16000
iteration:  18000
Final MSE:  0.012304761447
Linear Regression Duration:  599.0103874206543
>>>
```

1.3 Generalization:

- Once we apply SGD mini-batch with given batch-size, learning-rate and iteration parameters of (500, 0.005, and 20000) respectively, with varying weight-decay parameters, we observe the following:

Weight-Decay (λ)	Accuracy (Thresh. = 0.1)	Accuracy (Thresh. = 0.2)	Accuracy (Thresh. = 0.3)	Accuracy (Thresh. = 0.4)	Accuracy (Thresh. = 0.5)
0	0.67	0.83	0.92	0.97	0.99
0.001	0.67	0.82	0.91	0.97	0.99
0.1	0.71	0.83	0.91	0.92	0.97
1	0.70	0.83	0.90	0.93	0.95

- Note: varying thresholds were used, in give some idea as to how the data is scattered with respect to the hyper-plane of separation.
- Note: the way in which the threshold was used was that a demarcation region was created such that anything inside of 0+threshold <= m <= 1-threshold would automatically be classified as 'incorrect'. If the target was 0, and w*x < threshold, then the point was correctly classified. Similarly if the target was 1, and w*x > 1-threshold, then the point was again correctly classified.
- Above, in green, are the highest classification values for each threshold. The general consensus seems to be that as lambda increases (weight-decay increases), more points are correctly classified within a smaller threshold (ie. Less-margin of error for correctly classifying points). In addition, it is to be noted that for low lambda values, higher error tolerance means that larger weights are allowed (not penalized). Thus these weights allow for larger variation in values that can still be correctly classified, and thus this freedom results in more points being correctly classified at larger thresholds.

- From the table above, we seek to strike a balance between higher weight-decay and performance, and thus choose lambda = 0.1 as our weight-decay parameter of optimum choice.
- The test accuracy for this parameter is as follows:

Weight-Decay (λ)	**Accuracy (Thresh. = 0.1)**	**Accuracy (Thresh. = 0.2)**	**Accuracy (Thresh. = 0.3)**	**Accuracy (Thresh. = 0.4)**	**Accuracy (Thresh. = 0.5)**
0.1	0.717	0.848	0.910	0.931	0.951

- Observation: this test-set gave an accuracy largely comparable to the accuracy predicted by the validation-set, using a weight-decay parameter of 0.1.
- It is prudent to tune the weight-decay parameter on the validation set, rather than the training set, because the weights themselves are set in the training set. Therefore, setting the validation parameters based on 'accuracy on the training set' would be non-sensical, as the weights are meant to be a very close fit to the data on the training set. Instead, it would make sense to impose a check on allowable weights based on how well the model trained on the training data generalizes to the validation data. Thus, the validation set is used to tweak the weight decay coefficient.

1.4 Comparing SGD with Normal Equation:

- The following details were obtained while using the normal equation for linear regression:

Algorithm	Accuracy-List; Thresh = [0.1,0.2,0.3,0.4,0.5]	Training MSE	Cumulative Run-Time (seconds)
SGD minibatch with λ = 0	[0.67, 0.83, 0.92, 0.97, 0.99]	0.01426	116.73
Normal Equation	[0.66, 0.78, 0.89, 0.94, 0.97]	0.009392	1.43

- It is evident that the accuracy is very comparable between the normal equation and minibatch.
- In addition, Normal equation gives a lot less training MSE and in this case takes a lot less time.
- However, note that the computation of the matrix inverse is O(n^3). Thus if the number of dimensions of the problem (ie. Size of matrix) is increased (say 1024x1024 > 10^6 pixels in an image), then we would expect the Normal Equation to be computationally intractable on all but a super-computer. Thus, use SGD in this case!