Programming Project 3

Tanmayi Balla

November 6, 2022

Task-1

Implementation Overview

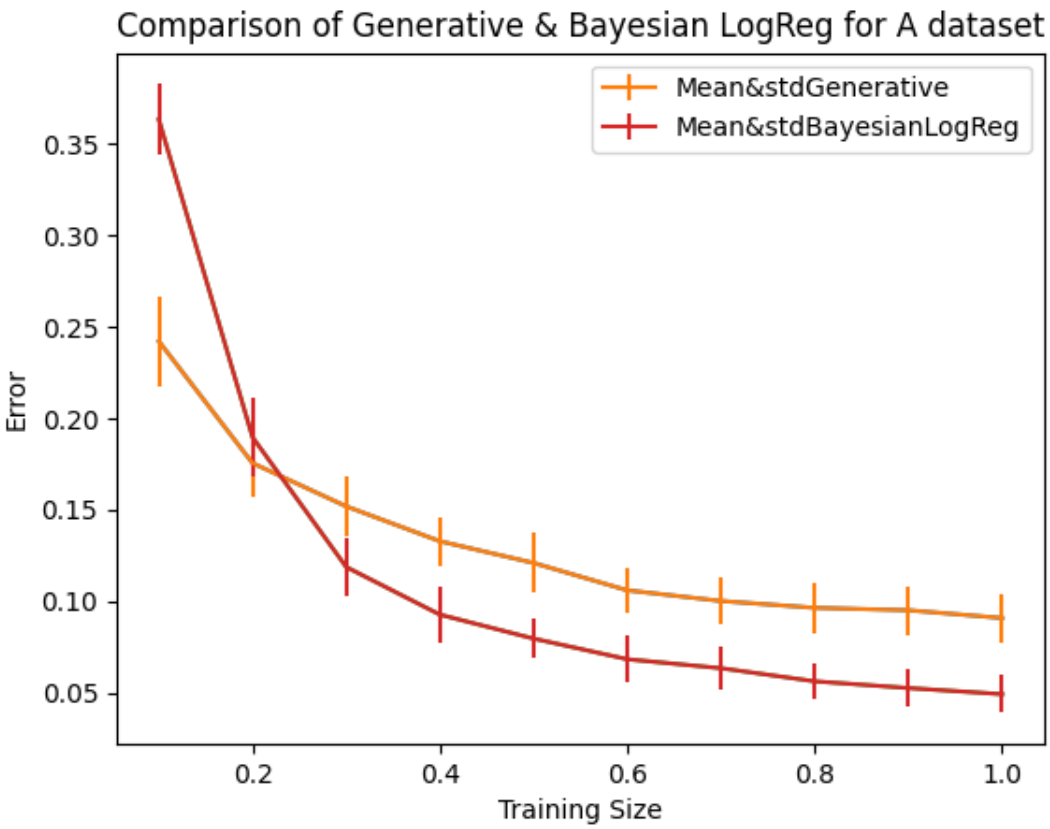
- The main objective of this task is to compare the performance of generative and discriminative models for different sizes of training set. For discriminative model, the algorithm used is Bayesian Logistic Regression.
- All the functions required for this task are created based on mathematical equations given.
- **Task1()** is the 1st function that is triggered, when we run Task1.py. It triggers the **Task1helper(X,Y)** function for each given dataset, i.e. A,B, and USPS, where X is the training set, and y is the labels.csv of that respective dataset.The function also stores the mean and std values returned in a result dataframe.
- **Task1helper(X,Y)** is the function that iterates over the algorithm 30 times, as asked in the question. It triggers **Task1A(X,Y)** and **Task1B(X,Y)** which return the error for 10 different training splits, for Generative and Bayesian Logistic Regression models. Task1B(X,Y) adds a feature with all ones to the dataset to represent W0, before triggering the BayesianLogisticRegression function.
- **GenerativeModel(trainX,trainY,testX,testY)** is the function that predicts the labels for the given dataset, and returns the error(difference between predicted labels and true labels). The function computes the mean1, mean2, S1, S2, S, W, and W0 for the given datasets using the equations from [B].
- **BayesianLogisticR(trainX,trainY,testX,testY)** is the function that predicts the labels for the given dataset, and returns the error. To compute the weights, this function triggers **computeweights(X,Y)** function, which computes weights using Newton's update equation.
- **traintestsplit(X,Y)** is used to randomly sample 2/3rd of the dataset for creating train and test datasets.
- Other helper functions such as **findmse()**, **sigmoid()**,**prediction()** (Used to classify the points into 0 and 1 based on the 0.5 threshold), etc. are used according to the requirement.

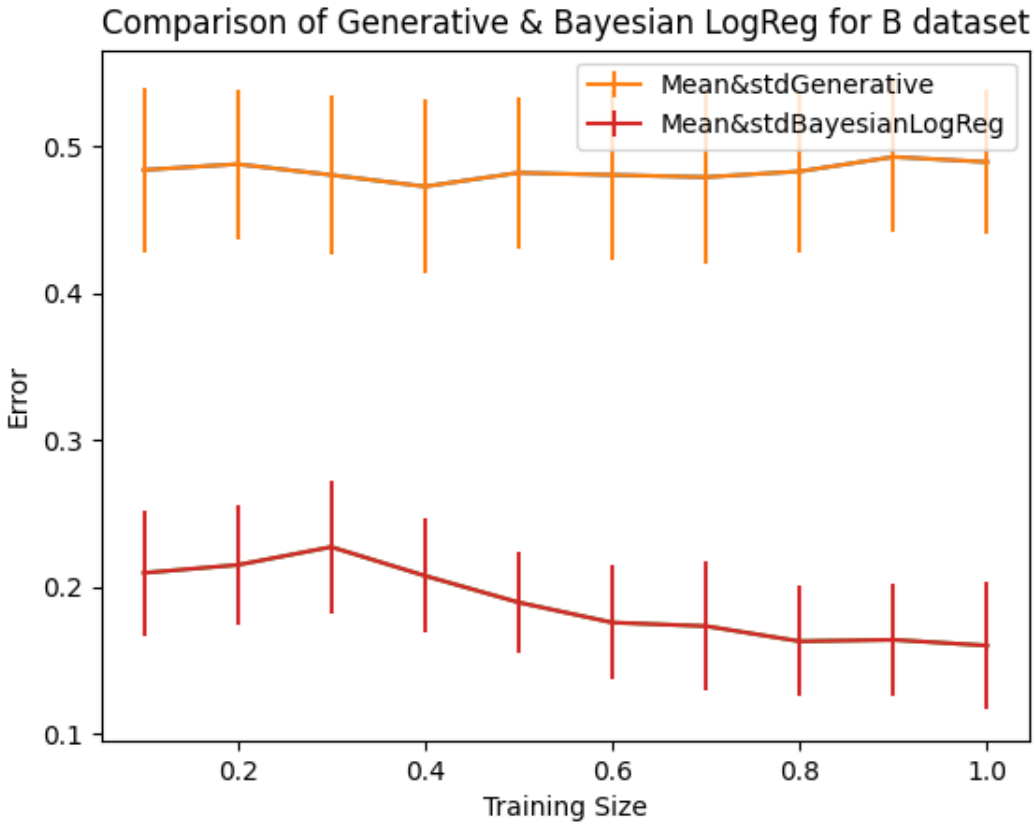
Results of Task-1

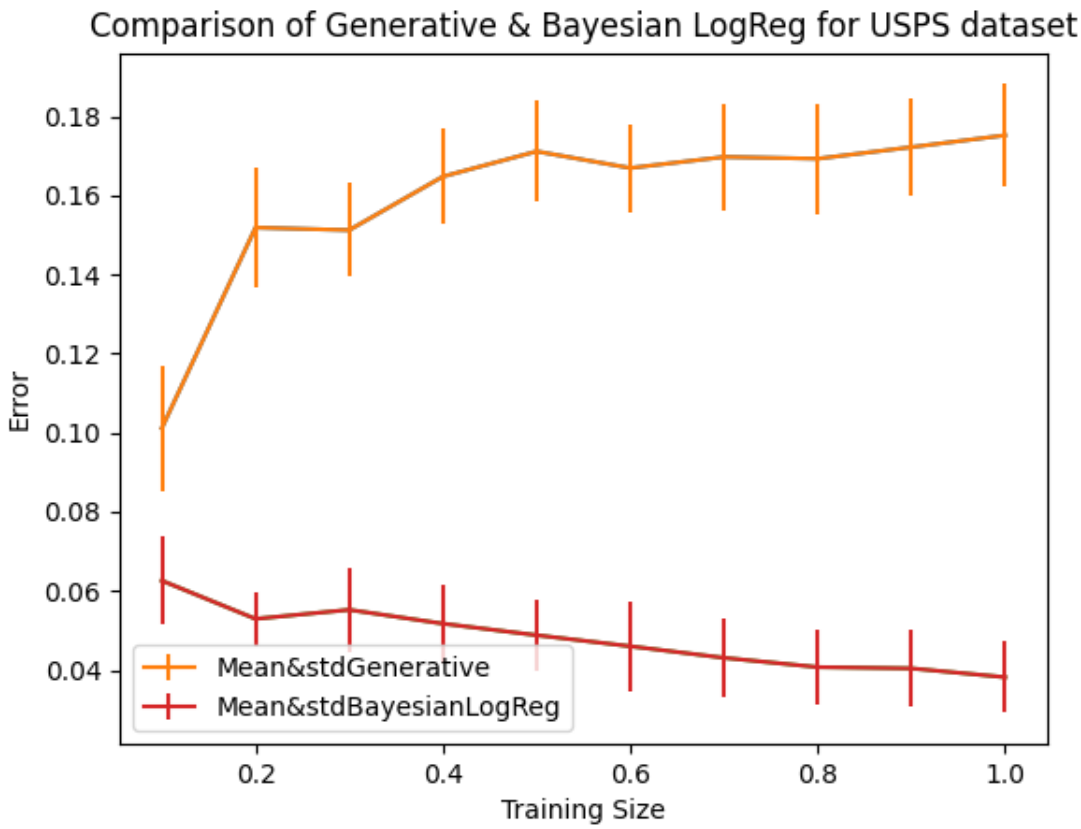
	training_size	A_MeanError_Generative	A_std_Generative	A_MeanError_Bayesian	A_std_Bayesian	B_MeanError_Generative	B_std_Generative
0	0.1	0.251569	0.0242801	0.366275	0.0190418	0.473529	0.0652605
1	0.2	0.170588	0.0204406	0.195931	0.0247935	0.476961	0.0479817
2	0.3	0.148529	0.0173961	0.121127	0.0134164	0.480882	0.051246
3	0.4	0.129755	0.0161081	0.0917647	0.0112805	0.47549	0.045775
4	0.5	0.118039	0.0158293	0.0786765	0.0111011	0.473039	0.046206
5	0.6	0.104902	0.0178394	0.0691667	0.0093355	0.476961	0.0462995
6	0.7	0.0996078	0.014212	0.0622059	0.009888	0.47598	0.0425342
7	0.8	0.0957843	0.0146702	0.057549	0.00828185	0.486275	0.0447557
8	0.9	0.0929412	0.0148978	0.0523039	0.00758158	0.489216	0.0420998
9	1	0.0883824	0.0126411	0.0496078	0.00859955	0.491176	0.0411765

B_MeanError_Bayesian	B_std_Bayesian	USPS_MeanError_Generative	USPS_std_Generative	USPS_MeanError_Bayesian	USPS_std_Bayesian
0.231373	0.064094	0.104198	0.0213286	0.0596692	0.0104054
0.230882	0.0489436	0.156361	0.0148964	0.0506361	0.00939755
0.25	0.0431261	0.153562	0.0135906	0.0517176	0.00718041
0.203922	0.0415483	0.166858	0.0135991	0.0480916	0.00781278
0.181373	0.0382227	0.172583	0.013969	0.0461832	0.00763995
0.177451	0.0434481	0.169529	0.0147007	0.0423664	0.00790546
0.175	0.0431846	0.17201	0.0121872	0.0386768	0.00811062
0.165686	0.0449165	0.171819	0.0118103	0.0374046	0.00654815
0.160294	0.0423417	0.174046	0.0125858	0.0366412	0.00652586
0.157353	0.0381881	0.177608	0.0129989	0.0358779	0.00676338

This Table contains the mean and standard deviation values, for datasets A,B, and USPS.


Comparison of Generative & Bayesian LogReg for A dataset
Mean&stdGenerative
Mean&stdBayesianLogReg
Error
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.2
0.4
0.6
0.8
1.0
Training Size


Comparison of Generative & Bayesian LogReg for B dataset
Mean&stdGenerative
Mean&stdBayesianLogReg
Error
0.5
0.4
0.3
0.2
0.1
0.2
0.4
0.6
0.8
1.0
Training Size



Inference

1. **A-dataset** Both the algorithms, Generative Model (GM) and Bayesian Logistic Regression (BLR) perform well on the "A dataset", since the data is uniformly distributed and linearly separable. Since, the data follow an underlying distribution, GM converges well, since it tries to capture this distribution and use conditional probabilities for classification. On the other hand, since this data is linearly separable, BLR also performed well, in terms of convergence. Both the algorithms reported low error rates for the dataset. Also, the standard deviation of the error is minimal for both the algorithms. With an increase in the train set size, both the algorithms reported a decrease in the error rate.

2. **B-dataset** The classes of this dataset have a different covariance structure, which makes the class densities, a quadratic function of x, leading to a quadratic decision boundary. Because of this, BLR performed much better than GM. There is no much convergence, with the increasing train set size, for both the algorithms. Almost similar error values are reported for all the training set sizes. BLR reported a bit higher error rates as compared to dataset A, owing to the quadratic decision boundary, and not completely linearly separable data. Also, the standard deviation for both the algorithms is higher as compared to the A-dataset.

3. **USPS-dataset** Since, this dataset deals with purely binary classification, BLR performed well as expected. GM gives higher error rate, with increase in the train set size, since the dataset might not have any underlying distribution, while the GM always tries to model using the class conditional probabilities (to find the underlying distribution). This is the reason why, GM performed well for

the least training size, as compared to the others. Higher standard deviations are reported for this dataset as well.

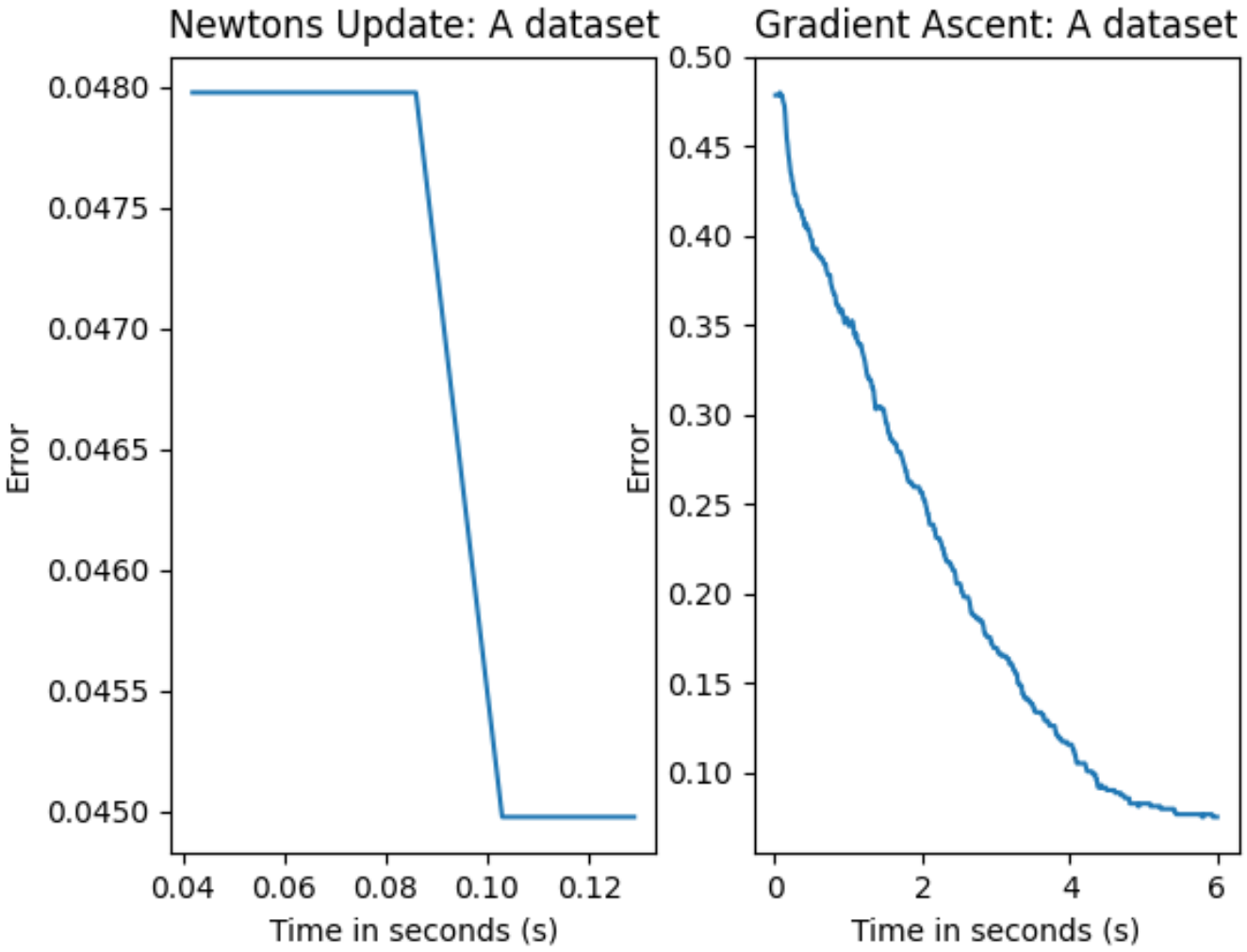
Task-2

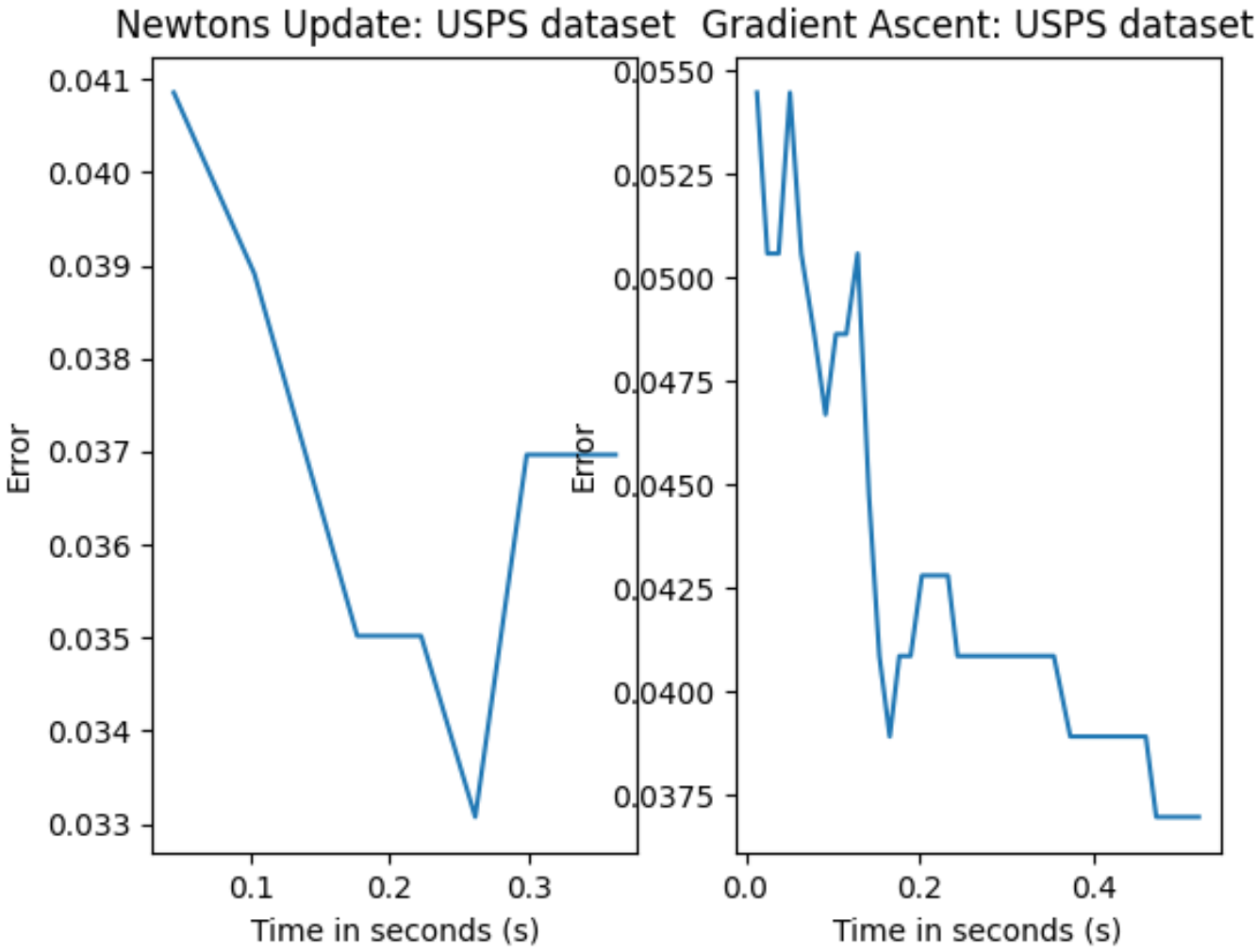
Implementation Overview

- The main objective of this task is to compare Newton's method to Gradient Ascent for generating weights for prediction.
- Two functions each are used for the Newton's updation and Gradient Ascent, which store the weights generated, along with the time taken to generate the same.
- For Newton's method, weights timestamps are stored for every update, while for Gradient Ascent, they are stored for every 10th update.

Results

```
Time taken for Newton's updates
[0.02, 0.04, 0.057, 0.072, 0.09]
Time taken for Gradient Ascent updates
[0.012, 0.022, 0.034, 0.046, 0.061, 0.073, 0.084, 0.096, 0.111, 0.124, 0.138, 0.151, 0.163, 0.175, 0.1
Error for Newton's updates
[0.047976011994003, 0.047976011994003, 0.047976011994003, 0.044977511244377814, 0.044977511244377814]
Error for Gradient Ascent updates
[0.4782608695652174, 0.4782608695652174, 0.4782608695652174, 0.4782608695652174, 0.4782608695652174, 0
```


Newtons Update: A dataset
Error
0.0480
0.0475
0.0470
0.0465
0.0460
0.0455
0.0450
0.04
0.06
0.08
0.10
0.12
Time in seconds (s)
Gradient Ascent: A dataset
Error
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
2
4
6
Time in seconds (s)



It is evident from these results that, Newton's method gives good accuracy, but is computationally expensive. This can be seen from the time taken in seconds in the above output. To generate the first weight vector (for the 1st update), Newton's method took over 0.02s. In the same duration, Gradient Ascent has generated almost 20 updates (each 10th update is stored in the list).

Though Gradient Ascent generates faster updates for the weight vector, it takes much time to converge, which is clearly evident from the graph above. For the 1st update, the error using weights from the Newton's method is way less as compared to the error using the weights from Gradient Ascent.

Both algorithms converged quickly, and reported lower error rates for "USPS dataset", as compared to the "A dataset".

Gradient Ascent took more time to converge for dataset-A, since the dataset contains uniformly distributed data.