REPORT: Logistic Regression

1. Introduction:

In this report, we try to analyse the performance (accuracy, fscore, recall and precision) of a logistic regression that we built using NumPy. The implementation of this network is similar to that of sklearn and the whole code has been written in a completely modularized fashion.

2. Data Preprocessing:

The dataset contained 1372 data points, each containing 4 input features (representing banknote features) and one binary target variable (representing if the note is authentic or not). In the preprocessing stage, all features were standardized and randomly split into *train_set*, *test_set* and *validation_set* (70-20-10 split ratio).

3. Non-Linear function:

- sigmoid: $y = \frac{1}{1+e^{-x}}$

4. Loss function:

- $L_n = \sum_{n=0}^{N} -t_n log(y_n) - (1 - t_n) log(1 - y_n)$

5. Model Architecture:

5.1 Vanilla Logistic Regression:

The architecture of our model resembles that of a single layer perceptron. The predictions for the whole training set were made according to the following formula: $\sigma(\widetilde{W}^T \widetilde{X})$ where $\widetilde{X}$ represents the input features padded with ones to accomodate the bias term and σ represents the sigmoid function. The Log Loss function mentioned above is a strictly convex function in the case of logistic regression and hence, only one global minima will exist. Any initiliazation of weights must converge at the global minima. *Gradient Descent* was used as the optimizing function in this case, i.e. the gradient with respect all training points was considered in order to make the "down hill step".

Top metrics achieved by the model:

- Best testing accuracy: 99.64%
- Best testing recall: 99.28%
- Best testing precision: 100.0%
- Best testing fscore: 99.64%

These metrics were achieved with 15000 epochs and a *learning_rate* of 0.01. In this case, we don't observe much difference in the

5.2 Logistic Regression with L1 regularization:

Regularization in Regression/Neural Network helps curb over-fitting. When given no restriction on the weights, the parameters can verly likely assume high values and overfit the training data. So, to control these weights from growing abnormally large, we introduce regularization. In L1 regularization, an additional term of $\frac{\lambda}{2} |W|$ is added to the loss function where λ is the regularization hyperparameter. In our model, the value of λ was searched between 0.0 and 1.5, and the value which gave the highest accuracy (or any metric, which could be explicitly specified) on the validation data, was chosen as the optimal L1 hyperparameter.

Top metrics achieved by the model:

- Best testing accuracy: 98.18%
- Best testing recall: 100.0%
- Best testing precision: 95.68%
- Best testing fscore: 97.79%

These metrics were achieved with 15000 epochs and a *learning_rate* of 0.01. In our case, not much difference in accuracy was observed even when L1 regularization was implemented. But the weights obtained initially were a bit a large and weights were slightly reduced due to the constraint. **Optimal lambda:** 0.107

5.3 Logistic Regression with L2 regularization:

This technique is the most popular regularization method and similar to L1 regularization. In this case, an extra term of $\frac{\lambda}{2} |W|^2$ is added to the loss function in order to curb higher weights and overfitting. Once again, the value of λ was searched between 0.0 and 1.5, and the value which gave the highest accuracy (or any metric, which could be explicitly specified) on the validation data, was chosen as the optimal L2 hyperparameter.

Top metrics achieved by the model:

- Best testing accuracy: 98.18%
- Best testing recall: 100.0%

- Best testing precision: 95.68%
- Best testing fscore: 97.79%

These metrics were achieved with 15000 epochs and a *learning_rate* of 0.01. In our case, not much difference was in accuracy was observed even when L2 regularization was implemented (values obtained were very similar to that of L1 regularization). But the weights obtained initially were a bit a large and weights were reduced due to the constraint. **Optimal lambda:** 0.107

6. Feature Importance:

One of the simplest options to get a feeling for the "influence" of a given parameter in a linear classification model (logistic being one of those), is to consider the magnitude of its coefficient times the standard deviation of the corresponding parameter in the data. Since our data is standardized, we just need to consider the magnitude of the weight. The weights (absolute values) obtained were: [10.97247675, 22.50767307, 25.61193349, 23.30568063, 1.4515536], where the first term denotes the bias. The largest coefficient is that of the second feature and the feature of least importance is fourth feature. Ranking the features by their importance, i.e. their magnitudes: [2, 3, 1, bias, 4]. Now if we look at the original weights then a negative coefficient means that higher value of the corresponding feature pushes the classification more towards the negative class. [Ref-1] [Ref-2]

7. Conclusion:

- The model almost everytime converged on the same point due the existence of a single global minima, and the initialization of weights didn't matter due to the same reason.
- Lower learning rates give a smoother curve (as seen below) but take more number of epochs to converge. Higher learning rates aid the model to converge faster but there is always a chance that the model might just end up bouncing around the minimia and never reach it, or might even "hop out" of the minima.
- Imposing regularization didn't affect the accuracy but the weights, in the case of L2 regularization, were significantly cut down.

[results on next page]

8. Results:

```
tp = 139, tn = 135, fp = 0, fn = 1
final accuracy: 99.63636363636364
final recall: 99.28571428571429
final precision: 100.0
final fscore: 99.64157706093191
weights:
 [[-10.07358672]
 [-20.37958812]
 [-22.38962402]
 [-20.63984824]
 [ -1.34369273]]
```

```
tp = 111, tn = 161, fp = 1, fn = 2
final accuracy: 98.9090909090909
final recall: 98.23008849557522
final precision: 99.10714285714286
final fscore: 98.66666666666667
weights:
 [[ -9.61124192]
 [-20.04918072]
 [-21.69429428]
 [-20.03327599]
 [ -0.09316027]]
```

Logistic Regression Metrics with no regularization

```
tp = 111, tn = 159, fp = 5, fn = 0
final accuracy: 98.18181818181819
final recall: 100.0
final precision: 95.6896551724138
final fscore: 97.79735682819383
weights:
 [[-10.97247675]
 [-22.50767307]
 [-25.61193349]
 [-23.30568063]
 [ -1.4515536 ]]
```

```
optimal L1 lambda: 0.8571428571428571
tp = 111, tn = 159, fp = 5, fn = 0
final accuracy: 98.18181818181819
final recall: 100.0
final precision: 95.6896551724138
final fscore: 97.79735682819383
[[3.07475239]
 [7.43789784]
 [8.56917647]
 [7.61481113]
 [0.01029291]]
```

```
optimal L2 lambda: 0.10714285714285714
tp = 111, tn = 159, fp = 5, fn = 0
final accuracy: 98.18181818181819
final recall: 100.0
final precision: 95.6896551724138
final fscore: 97.79735682819383
[[3.22482962]
 [7.66170758]
 [8.77604367]
 [7.82107333]
 [0.03278775]]
```

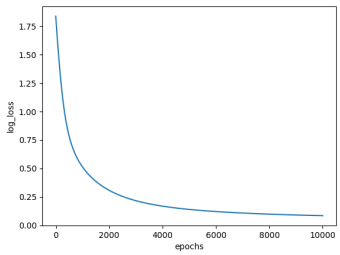
Logistic Regression Metrics and weights with no regularization, L1 regularization and L2 regularization wrt Accuracy

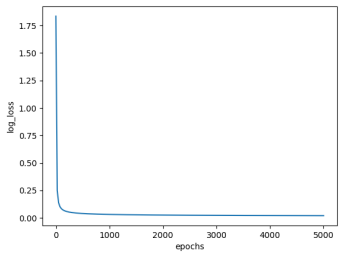
```
tp = 111, tn = 159, fp = 5, fn = 0
final accuracy: 98.18181818181819
final recall: 100.0
final precision: 95.6896551724138
final fscore: 97.79735682819383
weights:
 [[-10.97247675]
 [-22.50767307]
 [-25.61193349]
 [-23.30568063]
 [ -1.4515536 ]]
```

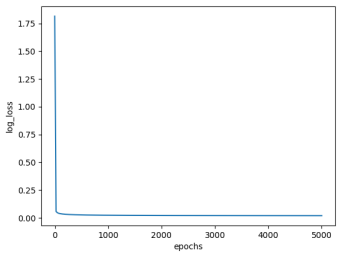
```
optimal L1 lambda: 0.8571428571428571
tp = 111, tn = 159, fp = 5, fn = 0
final accuracy: 98.18181818181819
final recall: 100.0
final precision: 95.6896551724138
final fscore: 97.79735682819383
[[3.07475239]
 [7.43789784]
 [8.56917647]
 [7.61481113]
 [0.01029291]]
```

```
optimal L2 lambda: 0.10714285714285714
tp = 111, tn = 159, fp = 5, fn = 0
final accuracy: 98.18181818181819
final recall: 100.0
final precision: 95.6896551724138
final fscore: 97.79735682819383
[[3.22482962]
 [7.66170758]
 [8.77604367]
 [7.82107333]
 [0.03278775]]
```

Logistic Regression Metrics and weights with no regularization, L1 regularization and L2 regularization wrt Fscore







Loss vs Epochs for different learning rate. $(10^{-5}, 10^{-3}, 10^{-2})$. We observe that lower learning rates give smoother curves.