ECE 6524: Deep Learning
Assignment 3: - Logistic Regression

Mridul Khurana (PID: mridul)

Introduction/Problem Statement

The problem statement is learning a Logistic Regression classifier. We have to create simulated data, build a custom logistic regression model and test how it performs compared with the scikit-learn library, and do hyperparameter tuning with learning rate.
In the end, discuss how to extend it for a multi-class classification problem as logistic regression natively is a binary classifier.
The results of each experiment are discussed at the experiment level

Method(s):

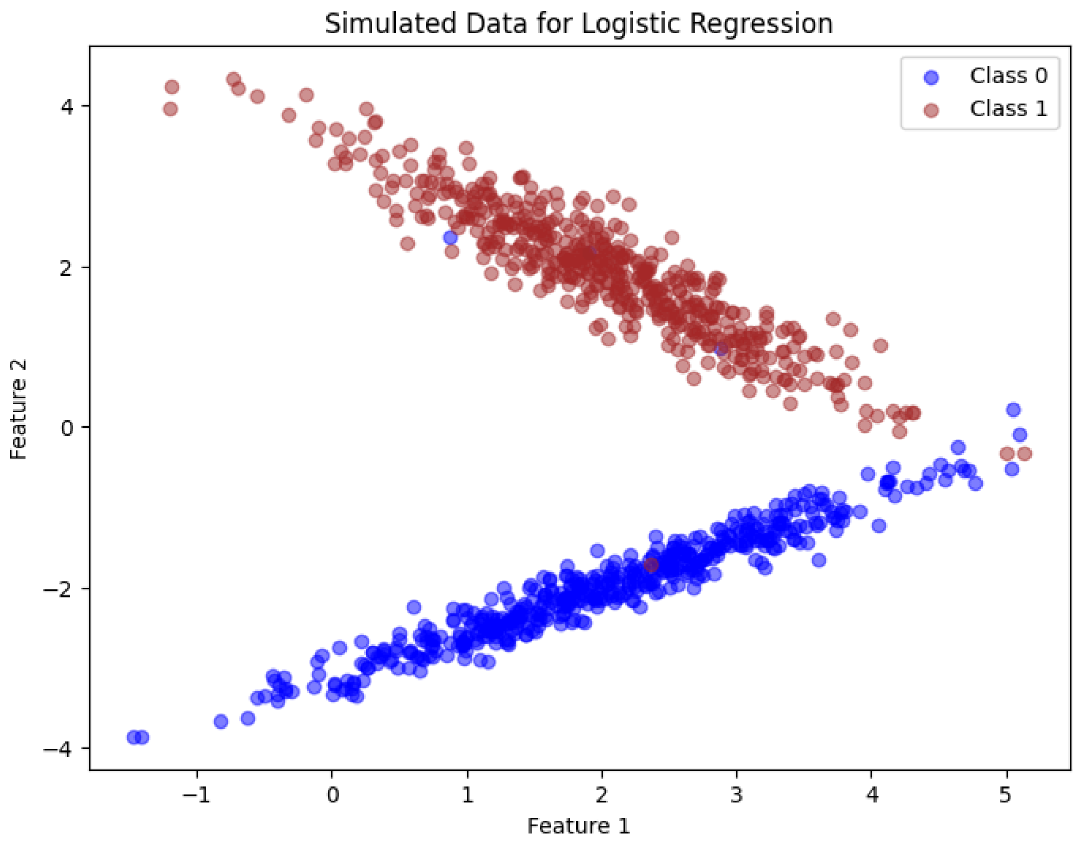
Question 2

(a) Generating Simulated Data

The figure in the assignment shows a binary classification simulated data. This can be achieved using the scikit-learn library using the function

from sklean.datasets import make_classification

Generated 100 samples with number of features as 2 as visualizing in 2d space is easier.

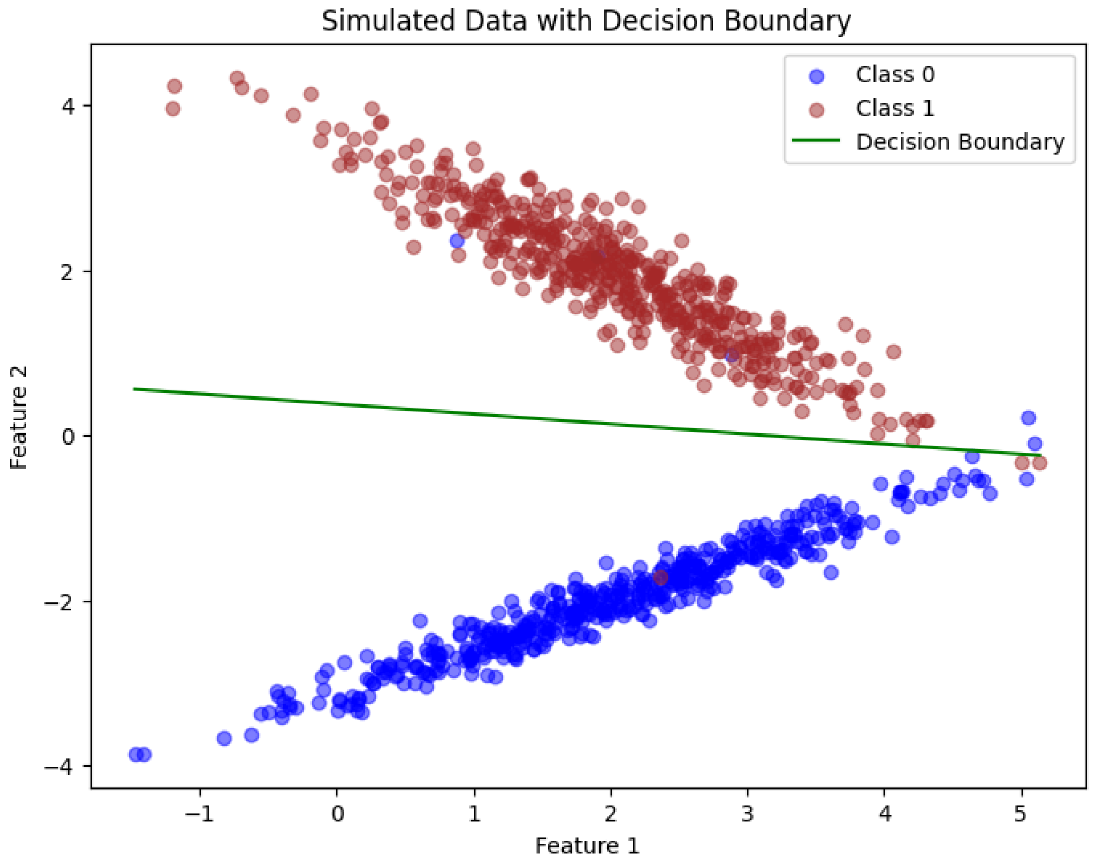


(b) Logistic regression Implementation

Built a logistic regression classifier using the scikit-learn learn library and used the *.fit()* function. With weights as :

Intercept: [-1.44267173]

Coefficient: [[0.45917037 3.78273318]]



Question 3

Implementation of custom SGD

- Implemented a custom function for logistic regression using Stochastic Gradient Descent (SGD)
- The loss function used was log-likelihood
- It takes the data, target labels, iterations and learning rate as input
- Trained it on the simulated data created in question 2.
- Comparing the weights of the custom Logistic Regression function with the scikit-learn implementation.
- The LR was set to 0.01 and iterations to 2000
- We can see that both the Models give almost similar results.

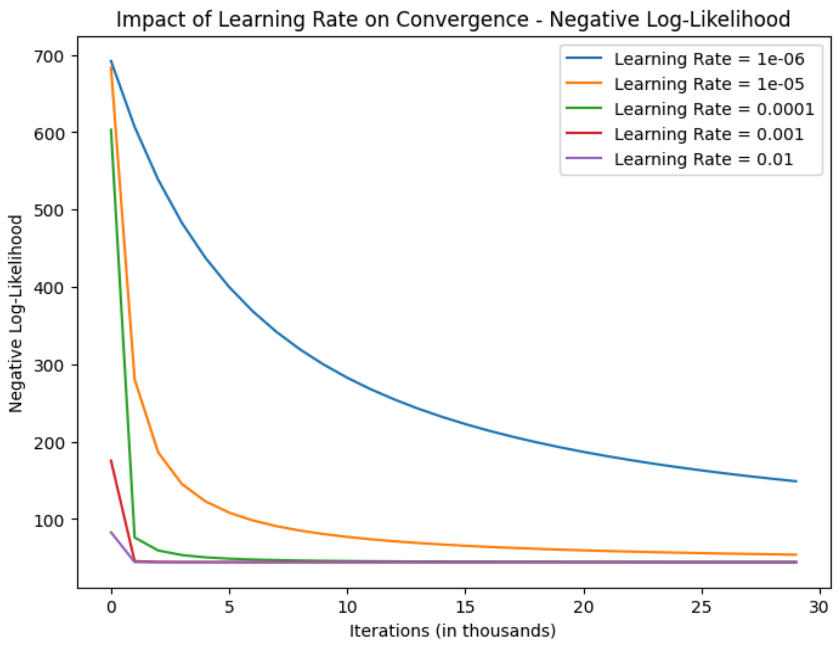
Weights	Scikit-learn implementation	Custom Logistic Regression
Intercept	-1.44267173	- 1.44262092
Coefficients	[0.45917037, 3.78273318]	[0.45915748, 3.78272752]

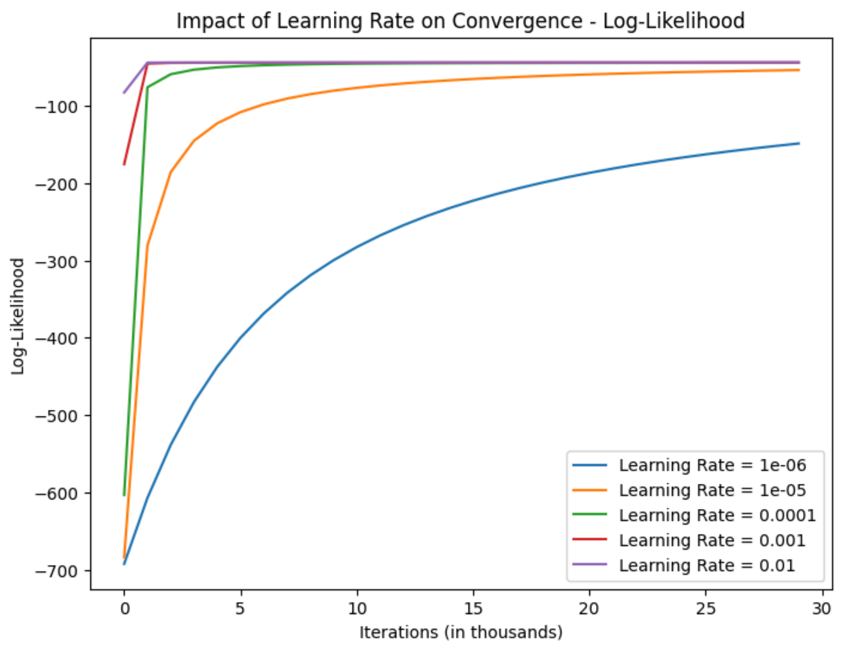
Experiments:

Question 4

Investigating the impact of Learning Rate on convergence of models

- Plotting the graph for Log-likelihood and Negative log-likelihood
- The learning rate used as [1e-6, 1e-5, 1e-4, 1e-3, 0.01]
- The graph is plotted across 3000 steps/iterations where the values are saved every 100 steps/iterations.
- **Discussion**: We see faster convergence for higher values of LR, but too high makes the model unstable





Question 5

Performance on the simulated data

- Split the simulated generated data into train and validation data
- Split is 70-30 with a random state of 22
- Used LR as 0.01, which performed best in the previous experiment
- **Discussion:** We see that the model fits nearly perfect with the simulated data

Train Data

Metric	Value
Accuracy	1.0
F1 Score	1.0
Precision	0.99
Recall	1.0

Validation Data

Metric	Value
Accuracy	0.98
F1 Score	0.98
Precision	0.98
Recall	0.99

Question 6

Multi-class dataset - IRIS

- I chose the IRIS multi-class dataset. It has 3 classes setosa, versicolor and virginica
- There are 150 images for each class.
- Here, I used the Standard Scaler function to bring the input features on the same scale. This is optional, but a good practice to ensure everything is on the same scale.
- The dataset is split with a 70-30 ratio and the same 3000 iterations with a LR of 0.01 is used
- **Discussion:** We can see that the model has fit perfectly for two sets of classes. This may be because the model can capture the discriminate features very well between the classes and the two classes are visually very different as well.
  For the classes, **Versicolor & Virginica,** the model seems to fairly well. This may be because the two classes have some similar features which may be harder to discriminate for every time, hence the drop in performance
  The overall performance of the classifier when averaged is fairly good for the simple dataset

Two Classes - Setosa & Versicolor

Metric	Value
Accuracy	1.0
F1 Score	1.0
Precision	1.0
Recall	1.0

Two Classes - Setosa & Virginica

Metric	Value
Accuracy	1.0
F1 Score	1.0
Precision	1.0
Recall	1.0

Two Classes - Versicolor & Virginica

Metric	Value
Accuracy	0.87
F1 Score	0.85
Precision	0.85
Recall	0.85

Overall/Average Performance of the classifier

Metric	Value
Accuracy	0.96
F1 Score	0.95
Precision	0.95
Recall	0.95

Question 7

Extending to Multi-Class Data with more than 2 classes.

In previous steps, we picked two classes and solved the problem as a binary classification. If we want to solve it as a multi-class problem there are a few other ways:

1. **One vs Rest (OvR) Strategy:**
    a. A separate logistic regression model is trained for each class to predict the probability that a given sample belongs to that class versus all other classes. This results in as many logistic regression models as there are classes. When classifying a new instance, the model that gives the highest probability is chosen as the prediction.
    b. Each binary classifier can be independently analyzed, which may provide insights into how each class is distinguished from the rest.
    c. This approach may lead to imbalanced datasets for each binary classifier, potentially affecting the performance.
2. **One vs One (OvO) Strategy:**
    a. involves training a logistic regression model for every pair of classes. If there are N classes, this results in N(N-1)/2 models, i.e, every model votes on which class the instance belongs to, and the class with the most votes is selected.

   b. It is computationally expensive as it requires a large number of models for datasets with many classes
   c. The voting scheme ensures that the class decision is made based on a majority

3. **Multinomial Logistic Regression (Softmax Regression)**
   a. This generalizes logistic regression to multiclass problems without having to train multiple binary classifiers. It uses the softmax function to squish the outputs of the linear model for each class into probabilities that sum up to 1. The class with the highest probability is chosen as the prediction.
   b. It is computationally efficient as only one model is trained, regardless of the number of classes, which can be more efficient than OvR or OvO in terms of both training time and model management.

Conclusion

To conclude, we implemented a custom logistic regression classifier using stochastic gradient descent (SGD) from scratch. We saw that our implementation matches the scikit-learn library weight matrix. This helps us validate our implementation.
Logistic regression works very well on simple datasets, which shows how a simple classification algorithm is powerful. We also say how smaller learning rates, slowed down the convergence of the classification algorithm and further discussed many strategies for enhancing the binary classifier to a multi-class setting.