CSoC IG Week 1 Report

Comparing NumPy and PyTorch Implementations

Advaith Biswas

May 29, 2025

1 Introduction

This report compares two implementations of a binary classification model: one using pure NumPy and the other using PyTorch. The goal is to evaluate and analyze the performance differences between these implementations in terms of accuracy, F1 score, PR AUC, confusion matrix, and memory usage.

2 Pure NumPy Implementation

The NumPy implementation of the model achieved relatively high accuracy but suffered from a very low F1 score and PR AUC, indicating poor recall or precision despite a seemingly good accuracy. This suggests a strong class imbalance or ineffective decision boundary.

Performance Metrics

- **Accuracy:** 0.8010
- **F1 Score:** 0.0140
- **PR AUC:** 0.2759
- **Memory Usage (MB):** 451.25

Confusion Matrix

$$\begin{bmatrix} 26512 & 50 \\ 6550 & 47 \end{bmatrix}$$

The confusion matrix highlights that while the model accurately predicted the majority class, it failed to effectively classify the minority class, leading to high false negatives.

3 PyTorch Implementation

The PyTorch implementation, while consuming significantly more memory, demonstrates improved performance in terms of the F1 score. This suggests a more balanced model performance even though the overall accuracy is lower than the NumPy version.

Performance Metrics

- **Accuracy:** 0.5606
- **F1 Score:** 0.3434
- **PR AUC:** 0.2645
- **Memory Usage (MB):** 2095.13

Confusion Matrix

$$\begin{bmatrix} 9852 & 7817 \\ 1897 & 2540 \end{bmatrix}$$

This confusion matrix demonstrates a better balance between precision and recall, despite an overall drop in accuracy. The model captures a significantly higher number of true positives compared to the NumPy implementation.

4 Analysis and Conclusion

While the NumPy model performs well in terms of accuracy, it fails in meaningful classification of the positive class. In contrast, the PyTorch model achieves a more acceptable F1 score by better handling the class imbalance. This trade-off between accuracy and balanced classification is crucial in applications where identifying the minority class is important. The increased memory usage in the PyTorch version is expected due to the framework overhead and the more refined model definition.

5 Use of LLM's

The use of LLM's in the projects are in the following places:

5.1 Pure NumPy implementation:

1.mounting Google Drive
2.in softmax function
3.in back propagation function
4.to one-hot encode 'Neighborhood' column
5.to calculate performance metrics

5.2 PyTorch implementation:

1.mounting Google Drive
2.in initializing 'class counts' and 'class weights' variables
3.in functions 'evaluate model' and 'get memory usage'