Prediction of Diabetes Using Perceptron Model

Aryan Puri
University of Adelaide
1914021

Abstract

The report presents a comparative study of various Perceptron algorithm variants for binary classification on the Pima Indians Diabetes Database. Five Perceptron variants are implemented and evaluated: Standard Perceptron, Perceptron with Momentum, Perceptron with Adaptive Learning Rate, Averaged Perceptron, and Mini-batch Perceptron. Performance is assessed using accuracy, precision, recall, and F1-score. The Averaged Perceptron demonstrates the best performance highlighting the importance of averaging weights to reduce overfitting. The results provide insights into the strengths and weaknesses of each variant. I also experimented with tuning the echoes to increase the accuracy.

1. Introduction

The project is an experiment to implement different variants of the Perceptron Model and compare their accuracy. Each variant has distinct modifications to improve performance or efficiency. The output of the model is Predicting Diabetes using data from a dataset of Pima Indians Diabetes Database using binary classification.

2. Perceptron Model and Developed Method

The model is used for binary classification tasks. It works by adjusting a set of weights applied to input features which help to minimize classification errors. The process is to iteratively optimize a set of random weights based on the difference between predicted and actual outcomes in the training dataset by using a learning rate to determine how much it should be updated during training. Next, the activation function determines the output of the model. I have used a sigmoid function to provide a probability-like output, which is thresholded to make binary predictions.

The Perceptron can be upgraded through various techniques, such as adding momentum to stabilize learning, adapting the learning rate over time, averaging weights across iterations to reduce overfitting, and using mini-batch updates for computational efficiency. Each variant offers unique advantages, e.g., the Averaged Perceptron often yields better generalization by smoothing out variations in weight updates. It is limited to linearly separable data. Experimenting different Perceptron variants on the dataset demonstrated varying levels of accuracy and performance, highlighting the importance of selecting appropriate hyperparameters and techniques tailored to the dataset.

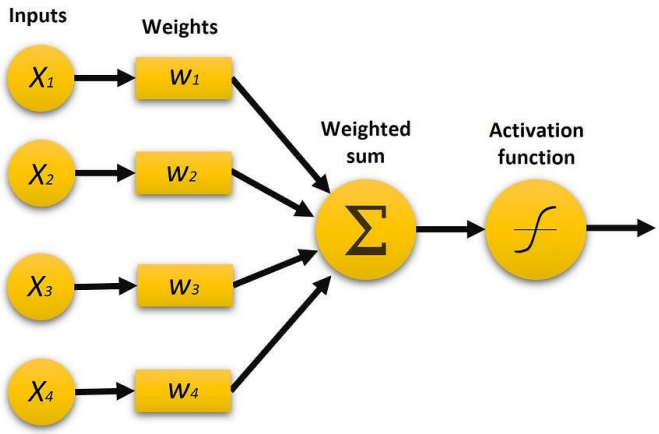


Fig. Components of a Perceptron.

3. Different Variations Explored

I have explored potential enhancements to the algorithm through various techniques, exploring their contributing to learning stability, efficiency, and generalization.

Methods I have explored are:

a) Momentum

Momentum technique uses the influence of previous weight updates helping smooth out the updates by adding a fraction of the previous update to the current one. This approach can potentially prevent model oscillation or getting stuck in local minima, facilitating faster convergence. The momentum term is controlled by a hyperparameter that determines how much of the past update is retained.

b) Adaptive Learning Rate

The adaptive learning rate technique involves dynamically adjusting the learning rate during training. Instead of using a fixed learning rate, it decreases over time, allowing for larger steps in the initial stages and smaller steps as convergence nears. This helps in avoiding overshooting the minimum and ensures finer adjustments towards the end of training, improving overall model accuracy and stability.

c) Averaging Weights

Averaging weights across iterations is an effective method to reduce overfitting. By averaging the weights obtained at each epoch, this technique smooths out fluctuations and noise in the learning process. It leads to better generalization on unseen data by preventing the model from fitting too closely to the training data.

d) Mini-batch Updates

Mini-batch updates enhance computational efficiency by processing subsets of the training data at a time, rather than updating weights for each individual sample or using the entire dataset at once. This balances between stochastic gradient descent (SGD) and batch gradient descent, providing faster convergence and reducing memory requirements. Mini batching also introduces noise into the gradient estimation, which can help escape local minima.

These enhancements contribute to more robustness and efficiency, allowing it to perform better. Experimenting with these techniques on the dataset has shown varying levels of improvement in accuracy and performance metrics, underscoring their importance in machine learning workflows.

I also explored impact of different epochs on accuracy of the model. An epoch is a complete pass through the entire training dataset. During each epoch, the model processes every data point in the training set once, updating its parameters based on the errors encountered.

How Epochs Affect Model Performance:

a) Learning Process: The number of epochs influences how many times the model sees the training data. More epochs allow the model to learn from the data multiple times, which can lead to better understanding and representation of the underlying patterns.

b) Overfitting: Increasing the number of epochs can lead to overfitting, where the model becomes too specialized to the training data and fails to generalize well to new, unseen data. This is because the model starts to memorize the training data rather than learning generalizable patterns.

c) Underfitting: Few epochs may result in underfitting, where the model does not have enough opportunities to learn from the data, leading to poor performance on both training and test sets.

d) Convergence: The optimal number of epochs is often when the model's performance on the validation set starts to plateau or degrade, indicating that it has learned as much as it can from the training data without overfitting.

4. Dataset

I used the scaled diabetes dataset at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html. Scaling boosts the efficiency and effectiveness of machine learning.

Scaling the dataset is crucial for several reasons:

a) Convergence Speed: Perceptron can be sensitive to feature scales. Features with larger ranges can disproportionately influence the weight updates, leading to slower convergence. Scaling ensures that each feature contributes equally to the distance calculations and weight updates, thereby speeding up convergence.

b) Model Performance: Unscaled data leads to models that perform poorly because the algorithm might interpret larger values as more significant, regardless of their actual importance.

c) Numerical Stability: Large feature values can lead to numerical instability in computations, especially when calculating gradients or performing matrix operations. Scaling helps maintain numerical stability by keeping values within a manageable range.

d) Interpretability: When features are on similar scales, it becomes easier to interpret the model coefficients and understand the relative importance of different features.

5. Implementation

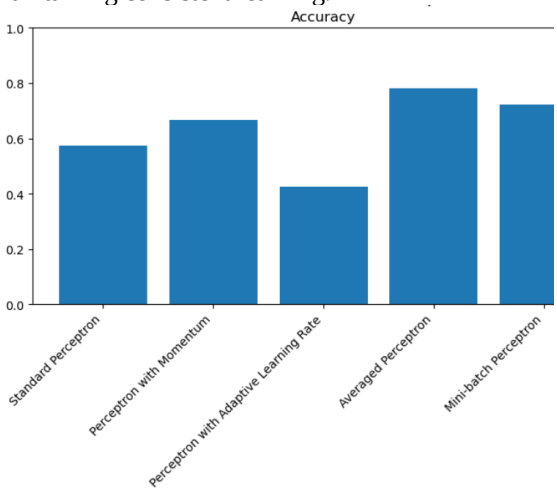
Please find the implementation at the following GitHub repo:
https://github.cs.adelaide.edu.au/a1914021/DL_Assignment1.git

6. Results

The evaluation of different Perceptron models on the Pima Indians Diabetes Database involved assessing several key performance metrics: accuracy, precision, recall, and F1-score. These metrics provide comprehensive insights into each model's effectiveness in binary classification tasks.

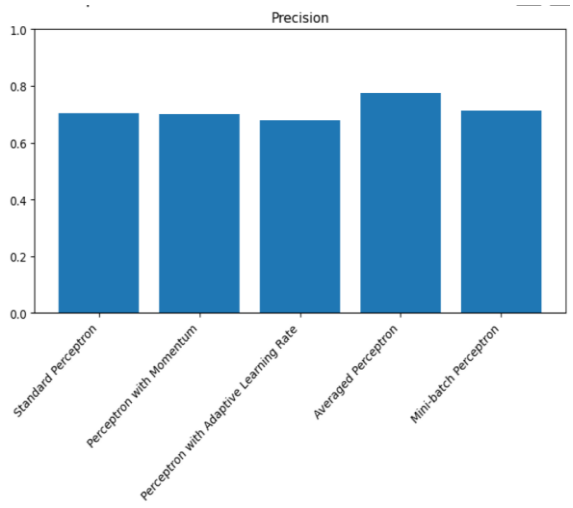
a) Accuracy

Accuracy measures the proportion of correctly classified instances out of the total instances. The Averaged Perceptron achieved the highest accuracy at 78.07%, indicating its superior ability to generalize across the dataset. The Mini-batch Perceptron followed with an accuracy of 72.37%, while the Standard Perceptron had a lower accuracy of 57.46%. The Perceptron with Adaptive Learning Rate performed the worst, with an accuracy of 42.54%, suggesting challenges in maintaining consistent learning.



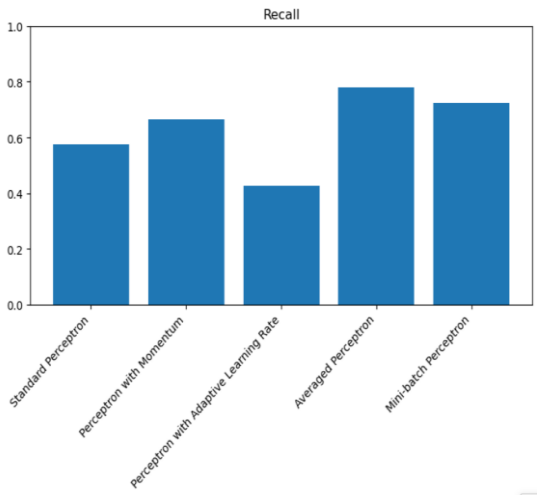
b) Precision

Precision indicates the proportion of true positive predictions among all positive predictions. The Averaged Perceptron again led with a precision of 77.65%, reflecting its capability to minimize false positives. The Standard Perceptron showed a precision of 70.25%, while the Mini-batch Perceptron had a precision of 71.42%.



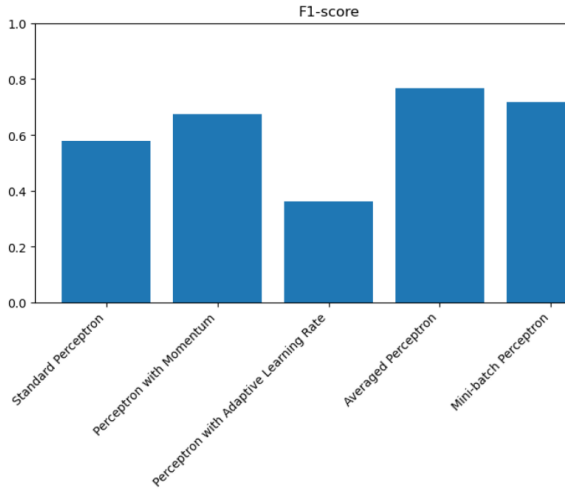
c) Recall

Recall, measures the ability to identify all relevant instances (true positives). The Averaged Perceptron excelled with a recall of 78.07%, followed by the Mini-batch Perceptron at 72.37%. The Standard Perceptron had a recall of 57.46%, indicating moderate effectiveness in capturing true positives.



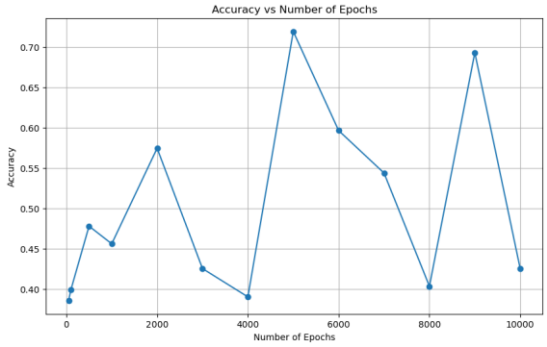
d) F1-score

The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. The Averaged Perceptron achieved an F1-score of 76.82%, demonstrating its balanced performance across precision and recall. The Mini-batch Perceptron had an F1-score of 71.64%, while the Standard Perceptron scored 57.75%.

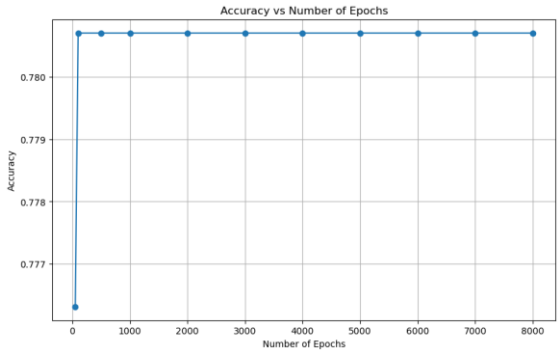


Epochs Comparison:

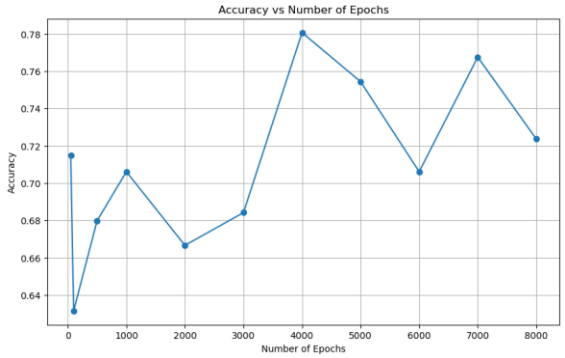
a) Standard_Perceptron:



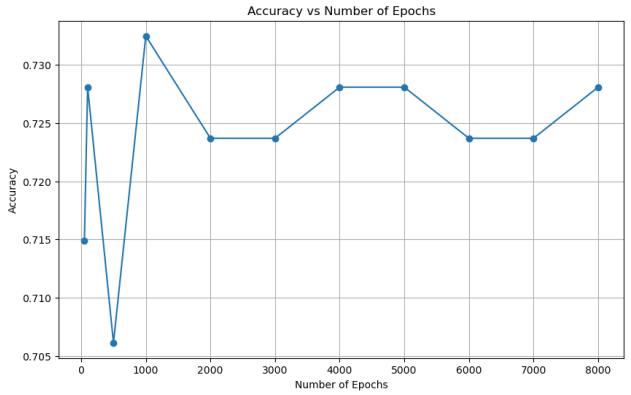
b) Averaged_Perceptron:



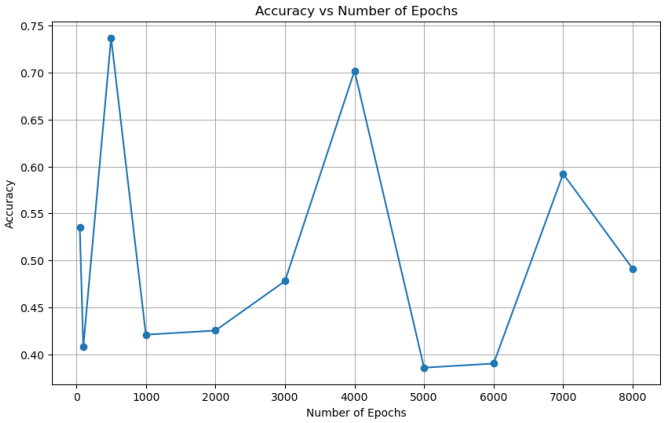
c) Perceptron With Momentum:



d) Perceptron with Minibatch:



e) Adaptive Perceptron:



Overall, these evaluations highlight the strengths and weaknesses of each model variant, emphasizing the importance of selecting appropriate techniques to enhance model performance on specific datasets. It also shows

References

[1] MarcusSena Building a Perceptron from Scratch: A Step-by-Step Guide with Python

[2] https://www.geeksforgeeks.org/what-is-perceptron-the-simplest-artificial-neural-network/
[3] https://www.youtube.com/watch?v=OFbnpY_k7js
[4] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html