Report on Implementing and Using the Perceptron for Digit Classification

1. Introduction

In this project, we implemented the perceptron algorithm to classify digits from the UCI machine learning handwritten digits dataset, also known as the **Digits dataset**. The perceptron is a fundamental linear classifier often used in machine learning for binary classification tasks. The aim was to better understand the perceptron algorithm and its application to a real-world classification problem. This involved both implementing the perceptron algorithm from a skeleton of a Python class and applying it to a multi-class classification problem.

The project was structured in two main parts:

1. Implementing the perceptron algorithm.
2. Applying the perceptron to classify digits from the Digits dataset.

2. Part I: Implementing the Perceptron Algorithm

The perceptron algorithm is based on a simple linear decision boundary. The model predicts the label of an input sample using a linear combination of its features, adjusting the weights during training based on misclassifications. The training process continues for a specified number of epochs over the dataset.

2.1. Perceptron Class Design

The perceptron was implemented as a Python class with two key methods: `train` and `predict`.

- **`train(X, y, epochs)`**: This method was designed to train the perceptron by iterating over the training data multiple times, adjusting the weights and bias whenever the model made incorrect predictions.

- **`predict(X_new)`**: This method predicts the labels for new samples by computing the sign of the weighted sum of the features.

During training, the perceptron updates its weights using the following rule when the predicted label is incorrect:

$w \leftarrow w + \alpha * yi * xi$

$b \leftarrow b + \alpha * yi$

where:

- w is the weight vector,
- xi is the input feature vector,
- yi is the true label,
- α is the learning rate.

The perceptron also includes a mechanism to handle cases where the learning rate is invalid (less than or equal to zero).

2.2. Model Validation

To verify our implementation, we tested the perceptron using a simple binary classification task (the AND logic gate) and compared the results with a scikit-learn implementation of the perceptron. The predictions, weights, and bias from our custom model matched those from the scikit-learn model, ensuring that the implementation was correct.

3. Part II: Using the Perceptron for Digit Classification

3.1. Dataset Overview

The **Digits dataset** from scikit-learn contains 8x8 pixel images of handwritten digits. These images are converted into 64-dimensional feature vectors. The task is to classify these images into one of 10 categories (digits 0 through 9).

3.2. Data Preprocessing

The dataset was split into three subsets:

1. **Training set** (60% of the data),
2. **Validation set** (20% of the data),
3. **Test set** (20% of the data).

We used `train_test_split` from scikit-learn to perform the data splitting. This ensured that the model was trained on one portion of the data, validated on another, and tested on a final unseen set of data.

3.3. Multi-Class Perceptron

The perceptron algorithm is inherently binary, but the Digits dataset requires multi-class classification (10 classes). To handle this, we implemented a **One-vs-All** strategy, where a separate perceptron was trained for each digit (class). During training, the labels for the current class were set to 1 (positive) and all others were set to -1 (negative). After training the individual perceptrons, the model predicts the class by selecting the one with the highest score.

3.4. Hyperparameter Tuning

We performed hyperparameter tuning to find the best configuration for the perceptron. We tested several learning rates and epoch values to determine the combination that resulted in the highest validation accuracy. The learning rates tested were 0.001, 0.01, and 0.1, and the number of epochs ranged from 10 to 100.

After training with various combinations of learning rates and epochs, we selected the best-performing configuration based on the validation set accuracy.

3.5. Model Training and Evaluation

Once the best hyperparameters were identified, the perceptron was retrained on the full training dataset using the selected configuration. The model was then evaluated on the test set, and the accuracy was calculated. We also generated a classification report, which provided detailed metrics like precision, recall, and F1-score for each digit class.

The final test accuracy achieved was approximately **93%**, indicating that the perceptron model performed well.

4. Results and Discussion

Using the best configuration (learning rate = 0.01, epochs = 50), the model was trained on the full training set and evaluated on the test set. The test accuracy achieved was 93%, which indicates that the perceptron classifier performed well on the unseen data.

The classification report for the test set provides additional insights into the performance of the model for each digit class. Here's a summary of the results:

Precision: The precision for most classes is high, especially for classes 3, 4, 5, and 7 (1.00). Recall: Recall is excellent for most classes, with perfect recall for classes 1, 2, 3, 4, 5, 6, and 7, meaning the model correctly identified almost all instances of those digits. F1-Score: The F1-scores are generally high, with the highest scores for digits like 3, 4, and 7. Overall, the model achieved an accuracy of 93% on the test set, with balanced precision and recall across the classes. The macro average and weighted average for precision, recall, and F1-score are all around 0.92 to 0.94, indicating robust performance across different classes.

5. Conclusion

In conclusion, this project helped us understand the perceptron algorithm and its limitations when applied to multi-class classification tasks. The perceptron model performed well for this multi-class digit classification problem, achieving a 93% test accuracy. The best results were obtained with a learning rate of 0.01 and 50 epochs. While the perceptron is a simple linear classifier, it shows that basic models can still achieve competitive performance on structured datasets like the Digits dataset. However, more advanced models such as multi-layer perceptrons or support vector machines may provide even better accuracy, especially when handling more complex datasets.

Acknowledgements

We would like to acknowledge the assistance of ChatGPT in providing guidance and suggestions during the implementation and testing of the perceptron algorithm. While the majority of the work was completed independently, some points of clarification and suggestions were particularly helpful in optimizing the code.

We worked together throughout the entire project, discussing and making decisions collaboratively at each stage. We regularly exchanged ideas, considered different solutions, and ensured we were aligned in our approach. This collaborative approach allowed us to complete the project effectively and in a balanced manner.