MALIS Project 3
Multi-Class AdaBoost (SAMME) with Perceptron Weak Learners

PAN Qizhi **BROWN Joel**

26. January 2025

1. Introduction

1.1. Overview

In this project, we built on our earlier digit-classification work by integrating a multi-class AdaBoost algorithm, known as **SAMME**, into our existing Perceptron jupyter project. Rather than relying on a single binary Perceptron per digit, we adopt a **combined** (one-vs-rest) Perceptron approach as the **weak learner**.

By iteratively adjusting sample weights for misclassified examples and combining the predictions of multiple boosting rounds, we enhance classification performance on the handwritten digits dataset (digits 0 through 9).

Leveraging **k-fold cross-validation**, we tuned our hyperparameters (number of boosting rounds (T), Perceptron learning rate (α_t), and training epochs). The final SAMME ensemble, trained with the optimal settings(T = 10, $\alpha_t = 0.1$, epoches = 100), achieves high accuracy (97.22%) and strong F1-scores (up to 0.97) on the test set—substantially exceeding the baseline established by individual Perceptron models.

This highlights the effectiveness of boosting and multi round reweighting in reducing errors across difficult-to-classify digit classes and shows the flexibility of AdaBoost in improving a linear classifier.

1.2. ChatGPT Usage

We used ChatGPT to accelerate our coding, assist and complete with refining our SAMME implementation. We originally conceived of the solution to implement SAMME with our combined perceptron as weak learners and used ChatGPT to help us write the model.

1.3. Contribution

Qizhi:

- Implemented SAMME with combined perceptron model as weak learner.
- Made all plots and analysed the figures

Joel:

- Implemented k-fold and metric analysis
- Analyzed ensemble performance

2. Perceptron Model Revised

From the last submission, we got two suggestions:

1. The training/validation/testing setup needs to be revised.
2. Can it happen that you get two ones from two classifiers?

For comment 1, we examined our model and found that there's no problem in our testing set split, but we kept splitting the training set and validation set in the training session. We revised it and make sure all the sets are well split at the beginning.

For comment 2, initially, we adopted a "first come, first serve" strategy, but we have now moved to a more robust tie-break based on each classifier's F1-score: whichever classifier (digit model) has a higher F1-score takes priority in claiming the sample. This better reflects the confidence of individual classifiers when conflicts arise.

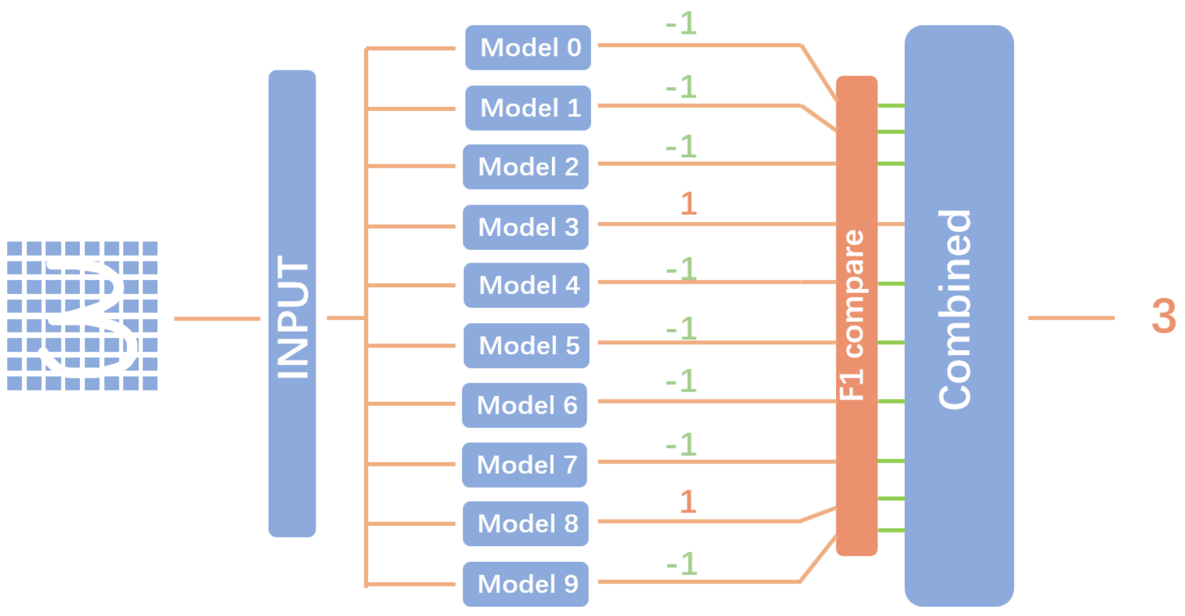


Figure 1: Revised combined perceptron model

3. Solution

3.1. SAMME Implementation

To implement SAMME, we need to determine the weak learner, how to relate the sample weights to the weak learner and how to update the weak learner weights.

3.1.1. Weak Learner

We use our combined perceptron model as the weak learner. The combined perceptron model itself is a multi-classifier consists of 10 simple perceptrons as binary classifiers, following one-vs-all strategy.

The parameters related to it are the learning rate **alpha** and training **epoches**.

3.1.2. Sample Weights Implement

One of the keys to successfully implement the SAMMES is to implement the sample weights correctly to the weak learners.

Sample weights affect the loss function, which further affect the weights and bias update rules in perceptron training. Here we implement them in update rules as:

$$w \leftarrow w + \alpha \cdot \omega_i \cdot y_i \cdot x_i$$

$$b \leftarrow b + \alpha \cdot \omega_i \cdot y_i$$

Here, ω_i is the weight assigned to the i-th sample, and α is the learning rate. w and b are the weights and bias in perceptron model.

3.1.3. Weak Learner Weights Update

After completely trained the t_{th} weak learner, we calculate its error as:

$$\text{Error}_t = \frac{\sum_{i:y_i \neq \widehat{y_i}} \omega_i}{\sum_{i=1}^{n} \omega_i}$$

The contribution of the t_{th} weak learner to the final model is determined by:

$$\alpha_t = \log\left(\frac{1 - \text{Error}_t}{\text{Error}_t}\right) + \log(n_{\text{classes}} - 1)$$

The sample weights ω_i are updated to emphasize misclassified samples:

$$\omega_i \leftarrow \omega_i \exp(\alpha_t \cdot 1(y_i \neq \hat{y_i}^t))$$

where

ω_i = Weight of the i_{th} sample.

y_i = True label of the i_{th} sample. $\hat{y_i}$ = Predicted label of the i_{th} sample.

n_{classes} = Total number of classes in the classification task. In our case, n_{classes} =10.

$1(y_i \neq \hat{y_i}^t)$ is an indicator function that equals 1 if the prediction is incorrect, and 0 otherwise.

α_t = Weight of the i_{th} weak learner.

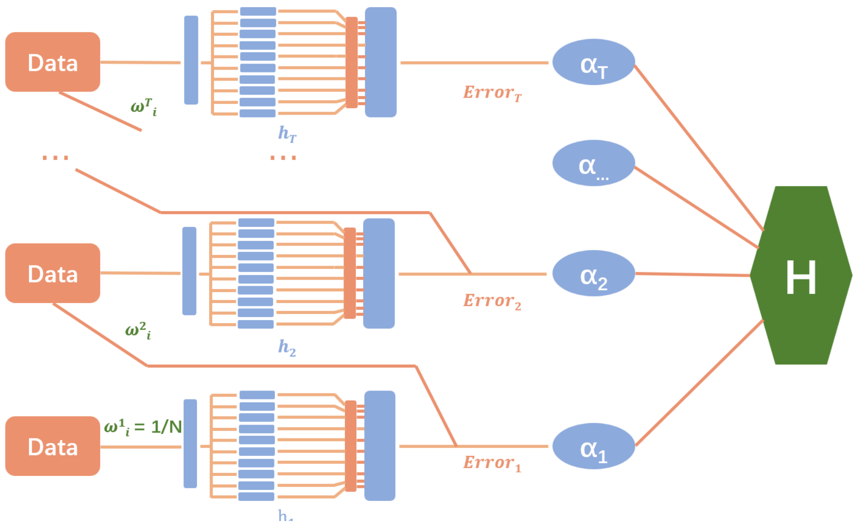


Figure 2: SAMMES implement illustration

3.2. Model Selection using K-Fold Cross Validation

3.2.1. K-Fold Cross Validation Implement

To determine the optimal hyperparameters for our multi-class AdaBoost (SAMME) model, we conducted a k-fold cross-validation process that incorporates all aspects of the pipeline —i.e., training T rounds of Perceptron-based boosting for each fold. This ensures the final hyperparameters we choose are robust to variations in our training data.

1. **Cross-Validation Setup**:
   - Define a small array of potential hyperparameters (e.g., `alpha ∈ {0.001, 0.005, 0.01}`)
   - Use k-fold (e.g., $\mathbf{k = 5}$) to split data into train/validation folds. Train the Perceptron on **k-1** folds and measure accuracy on the remaining fold.
2. **Averaging and Selection**:
   - For each `(alpha, epochs, T)` combination, compute the average validation accuracy over the **k** folds.
   - Pick the combination with the highest average accuracy (or F1-score).
3. **Applying the Best Hyperparameters**:
   - Use the "best" `(alpha, epochs, T)` from cross-validation in the SAMME loop. This ensures each weak learner Perceptron is reasonably tuned.

3.2.2. Data Splitting and Hyperparameters

- **Train/Validation/Test Split** = 8:1:1
- **Hyperparameters**:
  - Number of rounds (T) in SAMME: [3, 5, 10]
  - Perceptron learning rate (`alpha`) : [0.001, 0.005, 0.01, 0.1]
  - Perceptron training epochs : [10, 20, 30, 50, 80, 100]

3.2.3. Results and Model Selection

To notice, for learning rate = 0.2, we only test it for epochs=[50, 80, 100], which shows exactly the same performance as learning rate = 0.1 and learning rate = 0.005 shows exactly the same result as 0.01.

From the plots below we can see that, the performances for almost all the settings have obvious increase from epoch = 10 to epoch = 30. It means that for a quick, efficient and acceptable result, we could set the epoch to be 30. Epoch = 30 can be an elbow point for all other settings.

Increasing the number of weak learners T in SAMME training always improves the model performance. The monotonically increasing features of the model with T are also consistent with the design of the SAMME model. Trade off in this aspect is the model performance and the training cost.

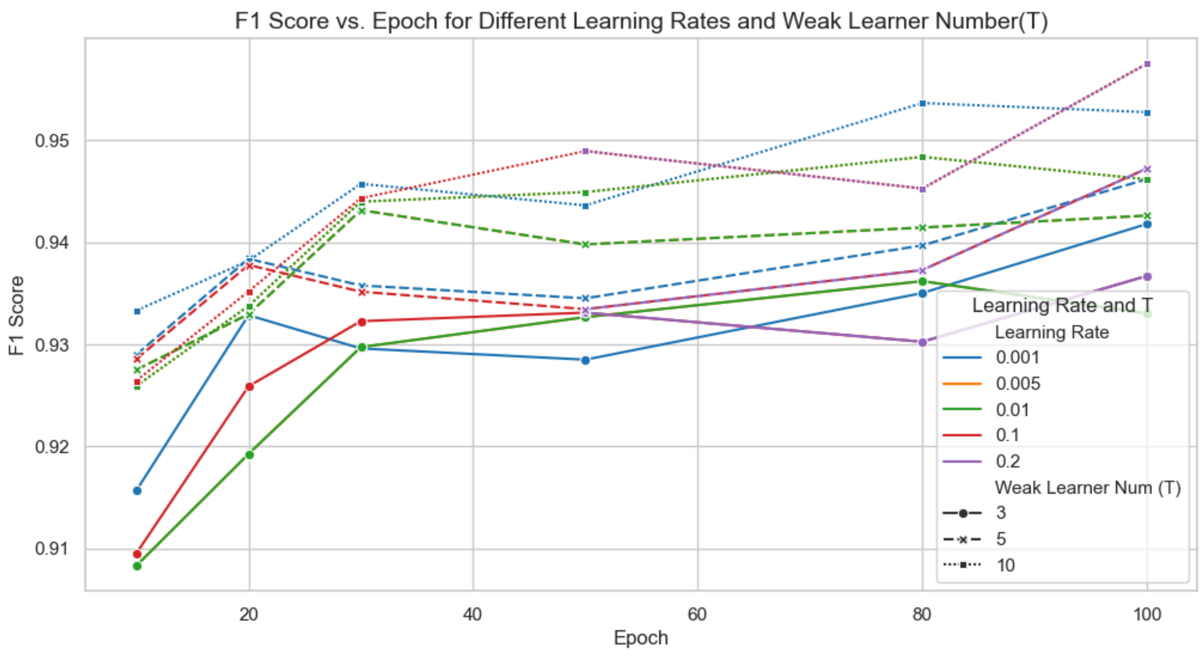


Figure 3: F1 score in K-fold cross validation

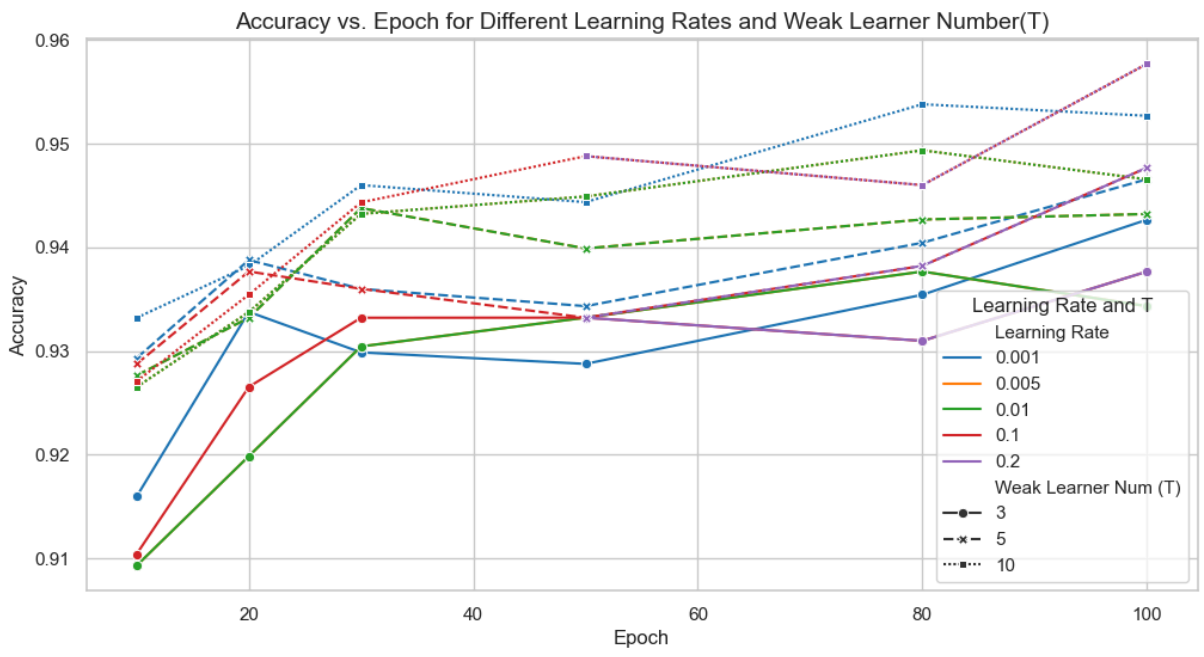


Figure 4: Accuracy in K-fold cross validation

In terms of the highest performance, we notice that (10, 0.1, 100) and (10, 0.2, 100) have exactly the same result. Concerning the training cost, we won't test the performance when T>10 and epoch>100. **We choose the (10, 0.1, 100) as the best parameters for the final testing**, because higher learning rate might show higher risk for non-converging.

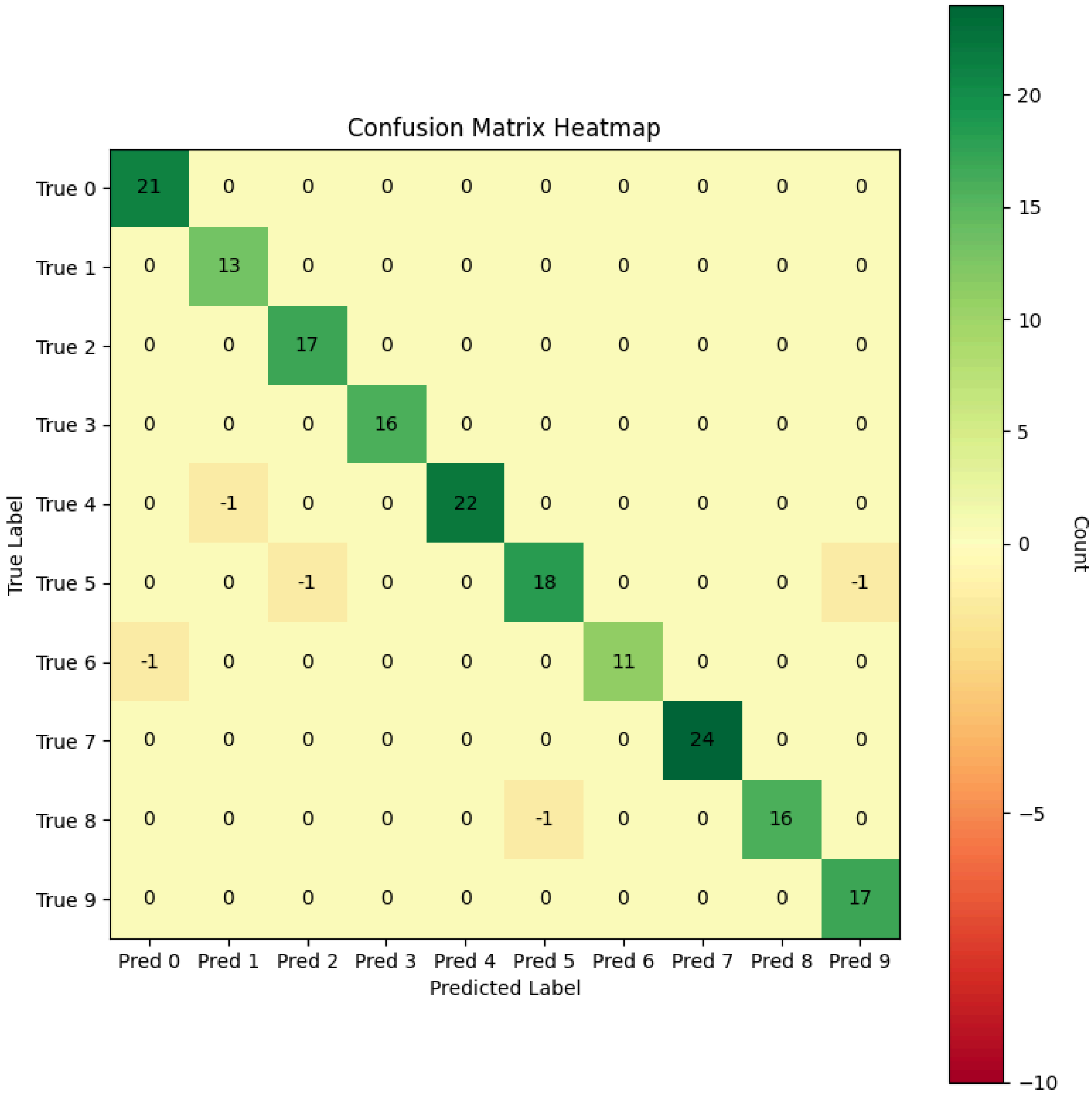
4. Best Model and Analysis

We select the best model with parameters (T = 10, learning rate = 0.1, epochs = 100).

The performance is shown below:

Metric	**Value**
Testing Accuracy	97.22%
Testing F1-score	0.97

Table 1: Best SAMME model performance



	Pred 0	Pred 1	Pred 2	Pred 3	Pred 4	Pred 5	Pred 6	Pred 7	Pred 8	Pred 9
True 0	21	0	0	0	0	0	0	0	0	0
True 1	0	13	0	0	0	0	0	0	0	0
True 2	0	0	17	0	0	0	0	0	0	0
True 3	0	0	0	16	0	0	0	0	0	0
True 4	0	-1	0	0	22	0	0	0	0	0
True 5	0	0	-1	0	0	18	0	0	0	-1
True 6	-1	0	0	0	0	0	11	0	0	0
True 7	0	0	0	0	0	0	0	24	0	0
True 8	0	0	0	0	0	-1	0	0	16	0
True 9	0	0	0	0	0	0	0	0	0	17

Figure 5: Best-parameter SAMME Model Results. Green areas mean the correct prediction and red areas mean the wrong prediction. The more counts, the deeper color.

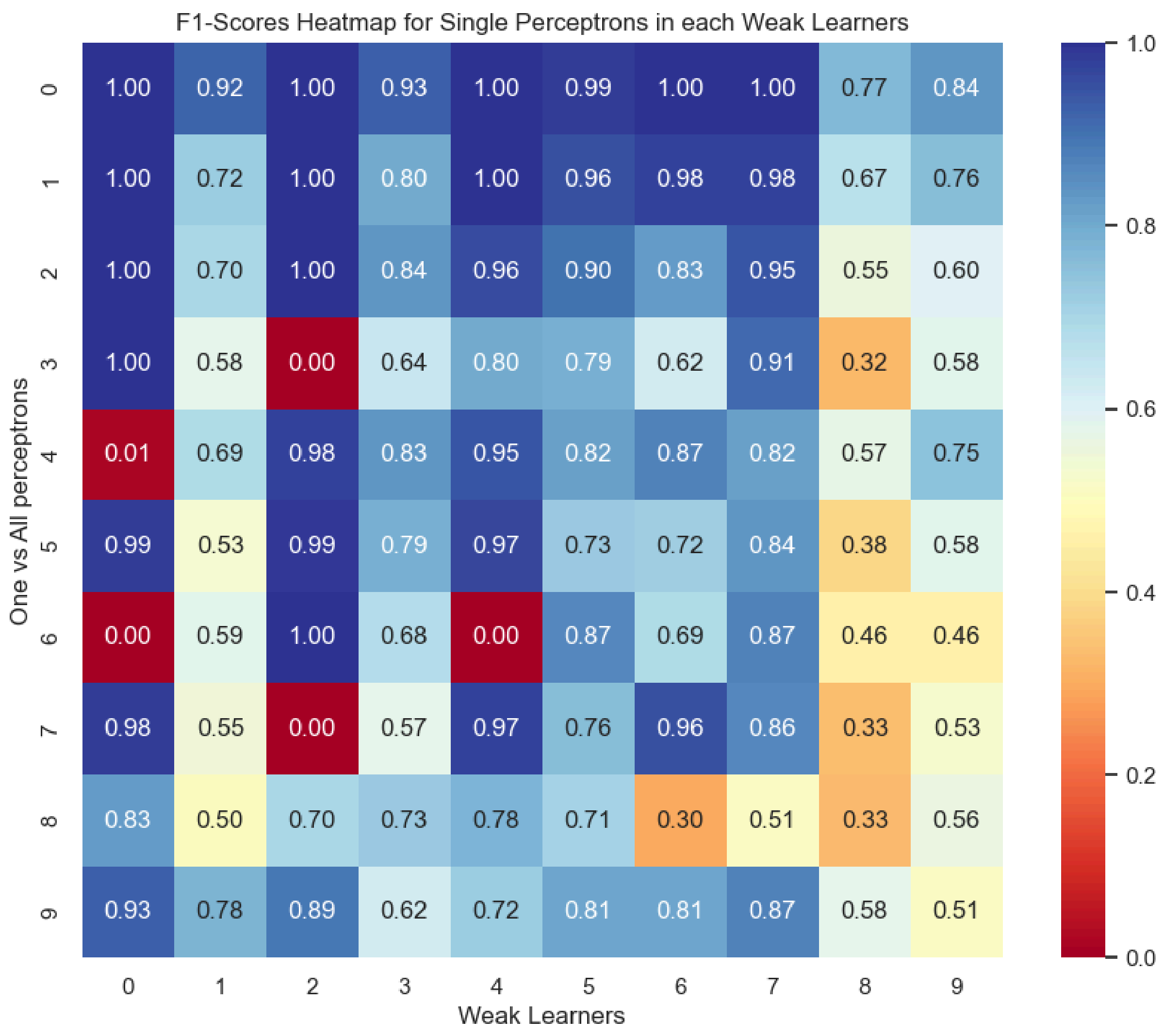


Figure 6: F1 score for each single perceptrons in one weak learner

This graph shows the f1 score of each simple perceptron inside a weak learner. Through this we can see how the sample weights change impacts the next round of learning.

For weak leaner 0, we can see the simple perceptron 4 and simple perceptron 6 have low performance in classifying digit 4 and digit 6. In the next round, due to the sample weights change, this two simple perceptrons pay more attention to the training and get better performance.

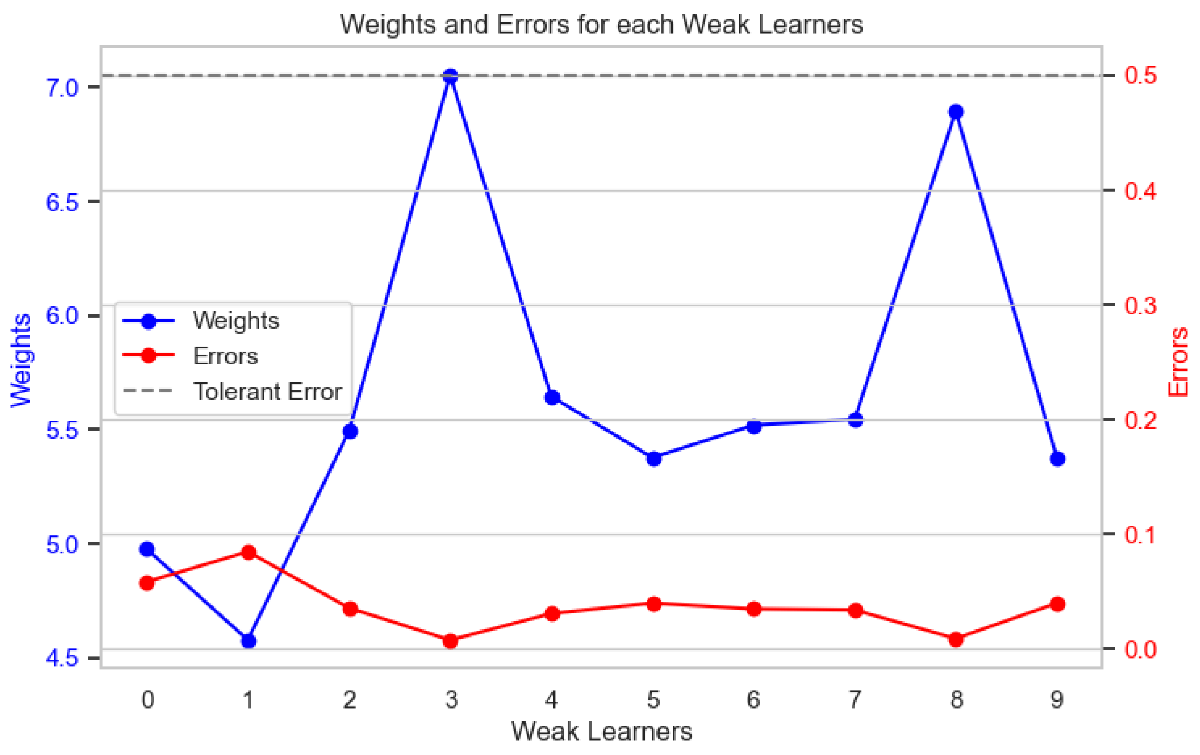


Figure 7: Performance of each weak learners

This graph shows the error each weak learner makes during the SAMME training and the weights allocated to it. We can see that the errors and the weights are approximately inversely proportional.

It's worth noticing that the weak learner 3 and weak learner 8 contribute most to the SAMME model. This good performance might attribute to all the single perceptrons they contain have good performance which can be examined from figure 6. This possible explanation works for weak learner 3 but is not quite suitable for weak learner 8. The specific reasons need further analysis.

5. Discussion

5.1. Observations on Performance

Our experiments demonstrated that increasing both the number of rounds (T) in SAMME and the Perceptron epochs leads to consistently improved accuracy and F1-scores. This aligns with the AdaBoost principle that boosting rounds help emphasize difficult examples, while additional training epochs refine each Perceptron's decision boundary. However, these improvements come at the cost of longer training time, indicating a trade-off between computational resources and marginal gains in performance.

In particular, the model with (alpha = 0.1), (text{epochs} = 100), and (T = 10) came out as our best settings, achieving around 97% accuracy and a 0.97 F1-score on the test set. Interestingly, higher learning rates (e.g., 0.2) showed similar performance in some runs but might risk some instability in convergence. Lower rates, however, required more epochs or additional boosting rounds to attain equivalent performance.

5.2. Weak Learners and Confusion Patterns

Our detailed analysis of single Perceptrons in each weak learner revealed that certain digits (like 4 and 6) initially was suffering from low F1-scores. Never the less, Some boosting rounds re-weighted these misclassified samples, allowing for later Perceptrons to focus on these problematic digit samples. This re-emphasis is key to AdaBoost's success and can be observed in the metrics where weak learner performance gradually converges for these difficult classes.

The confusion matrix and error metrics for each round show that digits (e.g., digits 8 and 9) remain challenging even with the boosted re-weighting. While error rates were generally low, future improvements—such as feature engineering or more sophisticated tie-breaking strategies could further reduce these confusions.

5.3. Practical Considerations

- **Model Complexity vs. Training Time**: While increasing (T) and epochs consistently improved performance, real-world scenarios might necessitate limiting these parameters for time-sensitive applications (eg. T=5 took 50 minutes to train, T=10 took 4 hours). .

Overall, our results confirm that **SAMME** with a carefully tuned Perceptron weak learner can achieve high accuracy and F1-scores on the digits dataset, with the re-weighting effectively alleviating hard misclassifications over multiple rounds.

6. Conclusion

In this project, we successfully applied **multi-class AdaBoost (SAMME)** with a **combined Perceptron** as the weak learner to classify handwritten digits. By first selecting the best Perceptron hyperparameters through **k-fold cross-validation** and then training (T) boosted rounds, we achieved strong performance on the digits dataset—reaching an accuracy of approximately **97.22%** on the the final test set, with corresponding F1-scores of **0.97**. Before boosting, we had an overall accuracy around 86.11% and F1-score near 0.87, indicating significant improvements over a single Perceptron baseline.

A. Appendix

- Project Repo: https://github.com/RizPur/MALIS-Project/tree/main/