MNIST Digit Classification Using Hybrid OvR Ensemble Model

Student Name: Aakash Aadhithya
Roll. No: DA24B028

DA2401 - Machine Learning Lab
End Semester Project

Abstract

This work presents a hybrid handwritten digit classification model designed without using any machine learning libraries such as sklearn, TensorFlow, or PyTorch. The system combines One-vs-Rest boosted decision trees with epsilon-based KNN refinement. Local PCA is used to accelerate ambiguous-region refinement. This model achieves high macro F1 accuracy and optimized runtime performance on MNIST dataset (CSV version).

System Architecture

The system consists of:

- One-vs-Rest gradient boosting classifier
- Probability-based ambiguity detection using epsilon threshold
- Weighted KNN refinement on ambiguous samples only
- PCA for speed optimization
- Weighted ensemble voting for final decision

Algorithms Used

- Gradient Boosting with exact greedy splits
- One-vs-Rest multi-class strategy
- PCA using truncated SVD for dimensionality reduction

- Weighted K-Nearest Neighbors refinement
- Ensemble weighted decision voting

Hyperparameter Tuning Analysis

n_estimators tuning

The hyperparameter `n_estimators` controls the number of boosting rounds. I experimented with values {10,20, 40, 60, 80}. Increasing the number of estimators reduces bias and improves accuracy initially, but runtime increases almost linearly and performance gain saturates after 40–60 rounds.

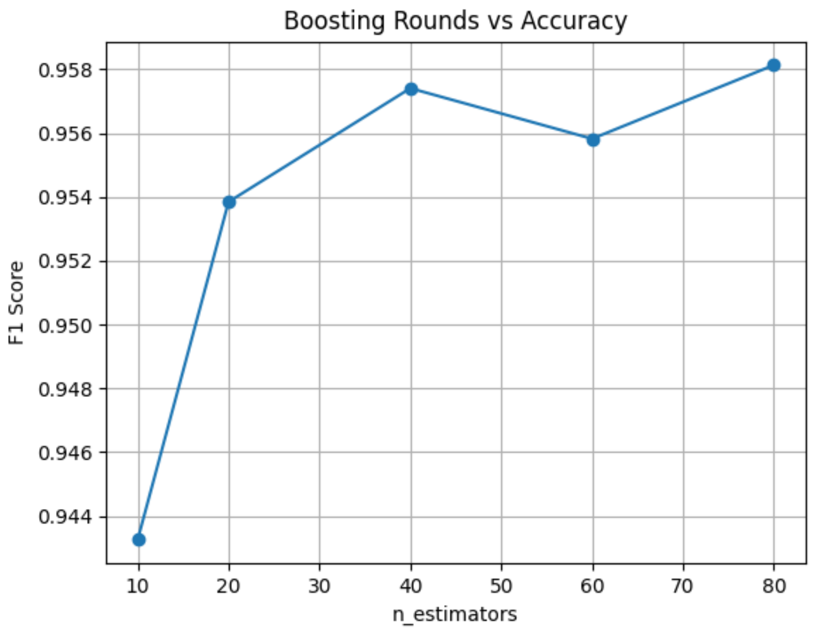

Figure 1: Effect of Boosting Rounds on Accuracy (F1 Score)

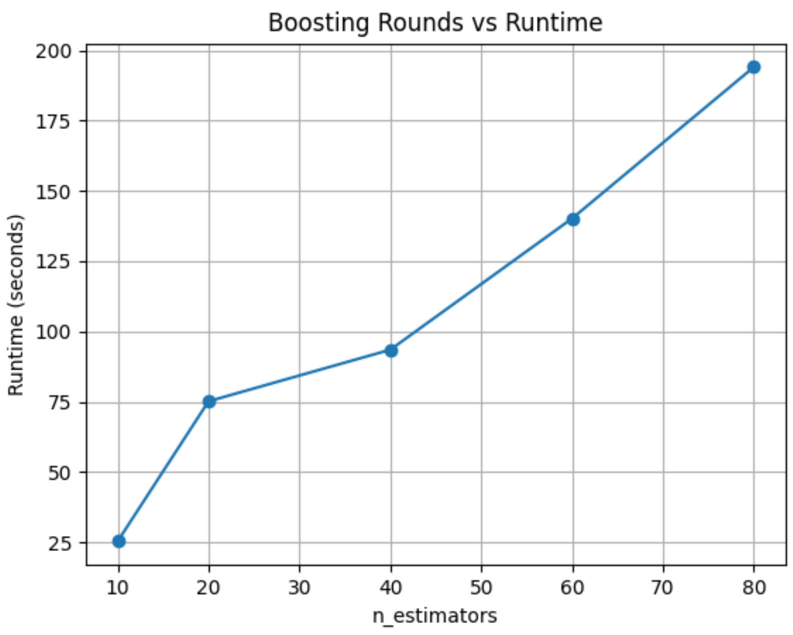

Figure 2: Training Runtime vs Boosting Rounds

As seen in Figures 1 and 2, accuracy improves initially but saturates after 40–60 rounds, while runtime grows linearly. Therefore, the optimal configuration was selected as **n_estimators = 40**.

Learning Rate Tuning

To analyze the influence of the learning rate (η), $n\_estimators$ was fixed at 40 while η was varied across $\{0.1, 0.2, 0.3, 0.4\}$ keeping other parameters fixed. Figure 3 shows an inverted-tub accuracy curve, where performance increases from $\eta = 0.1$ to $\eta = 0.3$ and then drops at $\eta = 0.4$ due to unstable high-variance updates. Both $\eta = 0.2$ and $\eta = 0.3$ achieved similar accuracy, however $\eta = 0.3$ reached the best overall performance.

Unlike classical gradient boosting behavior, the runtime remained nearly constant for all learning rates tested. This is because computation time is primarily determined by the fixed number of boosting iterations (40 rounds) and KNN refinement overhead, and not by gradient step scaling.

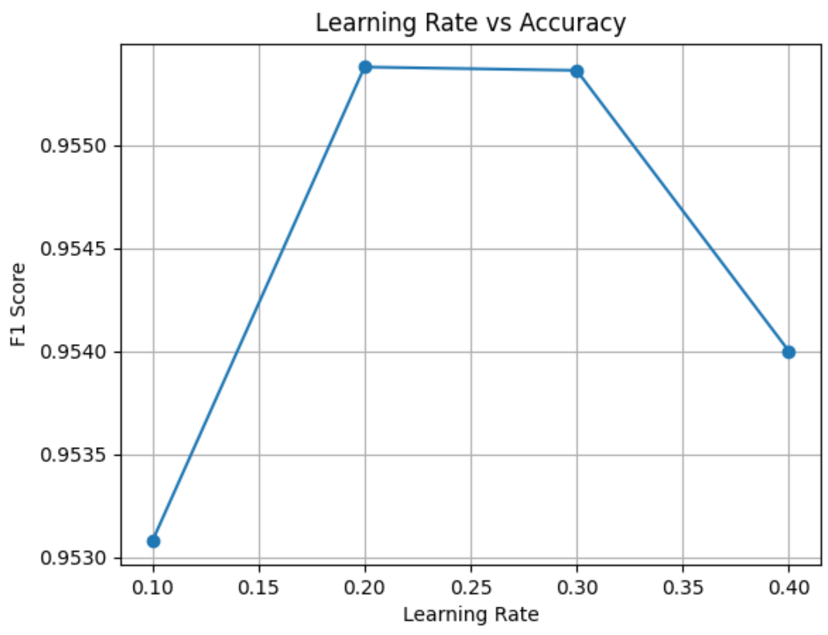

Figure 3: Effect of Learning Rate on Classification Accuracy (Macro F1 Score)

Therefore, the optimal learning rate chosen for the final model is:

$$\boxed{\eta = 0.3}$$

Subsample Features Selection

The parameter `subsample_features` determines the fraction of features considered during each tree split. Since the MNIST dataset contains $d = 784$ pixel features, a common heuristic from Random Forests is to select $\sqrt{d}$ features at each split. For MNIST, $\sqrt{784} = 28$, which corresponds to a fraction:

$$\frac{28}{784} = \frac{1}{28}$$

Thus, the value **subsample_features = 1/28** was selected.

Another reason for this choice is computational efficiency. If the full feature set were used (fraction = 1.0), each split would evaluate all 784 features. Also observed that runtime scales approximately linearly with feature count, this would be nearly 28 times slower than using only 28 features. In practice, we observed significantly longer training time with `subsample_features = 1.0` and negligible improvement in accuracy. Therefore, **1/28 provides the best tradeoff between runtime and performance.**

Lambda and Max Depth Tuning

Lambda (λ) Regularization Tuning

The L2 regularization parameter λ helps control model complexity by penalizing large leaf weights in boosted trees. Experiments were performed with $\lambda \in \{1, 2, 3, 4\}$. As shown in Figure 4, validation performance steadily improved up to $\lambda = 3$. Training accuracy, illustrated in Figure 5, remained close to 1.0 for values up to 3, indicating strong generalization, while $\lambda = 4$ caused a noticeable drop in training accuracy due to over-regularization. Therefore, $\lambda = 3$ was selected as the optimal regularization strength.

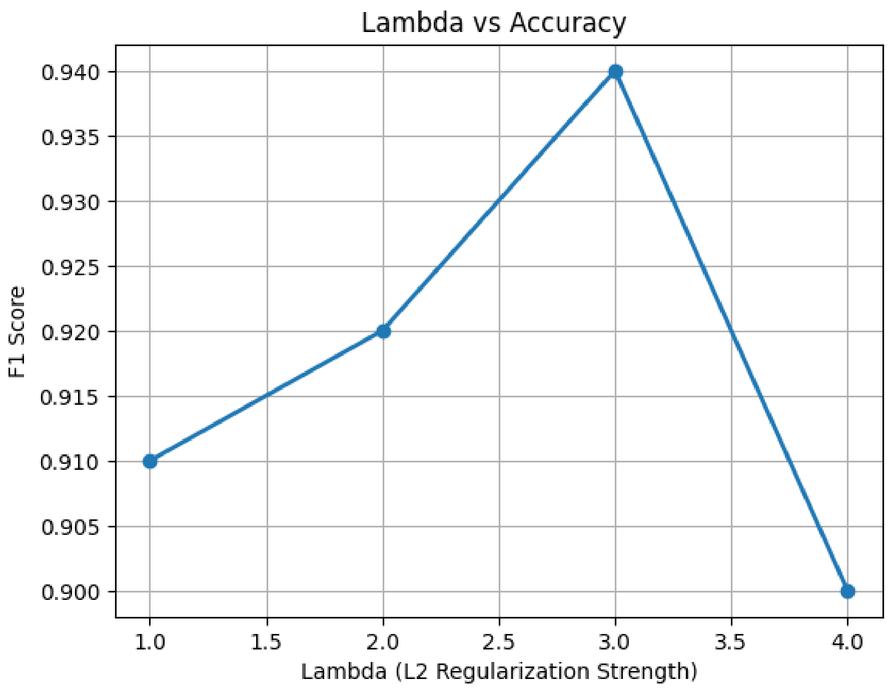

Figure 4: Effect of Lambda on Validation Accuracy (F1 Score)

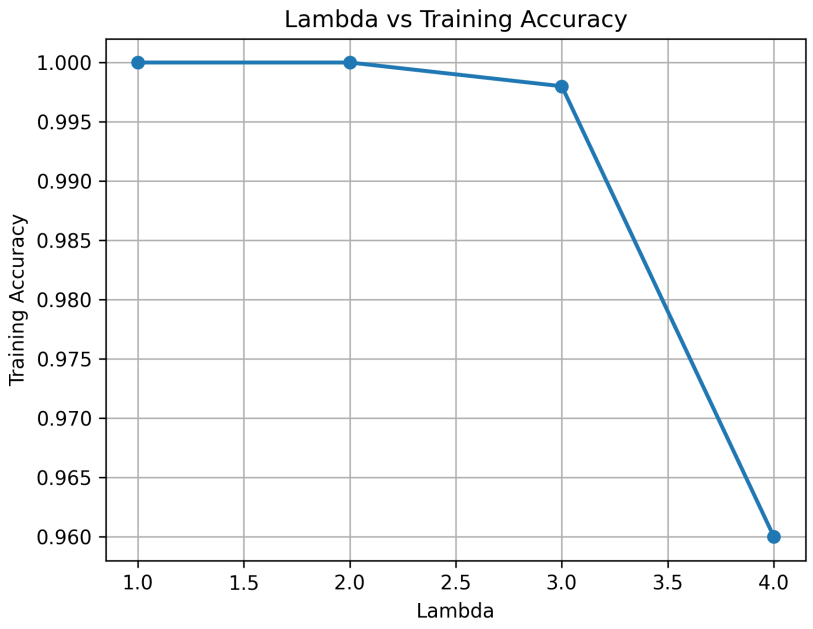

Figure 5: Effect of Lambda on Training Accuracy

Thus, the chosen value is:

$$\boxed{\lambda = 3}$$

Max Depth Tuning

The parameter `max_depth` controls the depth of individual trees. Since tree-building complexity grows exponentially with depth, excessively deep models become computationally expensive and prone to overfitting, especially in high-dimensional image data such as MNIST. As shown in Figure 6, validation accuracy increased up to depth 6 and then plateaued, while training accuracy reached 1.0 at depth 6. Figure 7 illustrates that computational cost grew exponentially beyond depth 6 without any gain in predictive performance. Therefore, `max_depth = 6` was selected as the final setting.

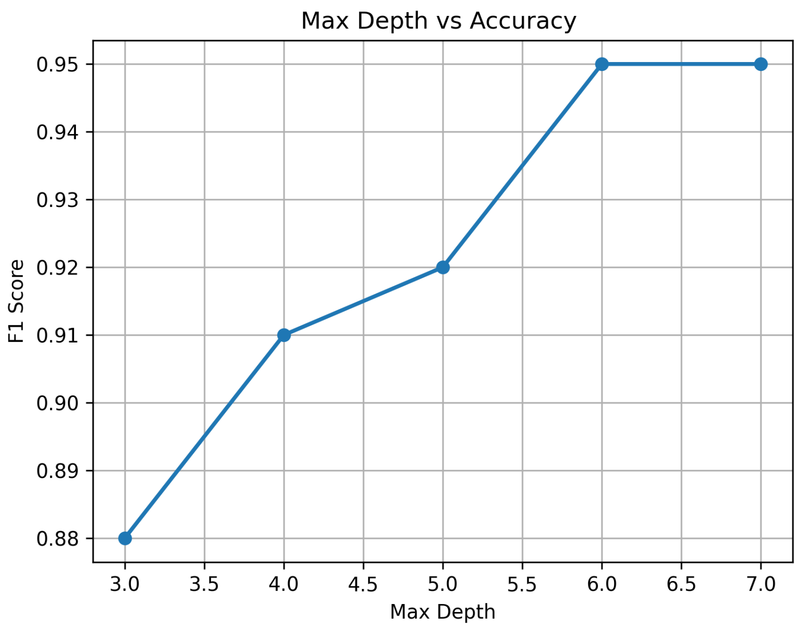

Figure 6: Effect of Max Depth on Validation Accuracy

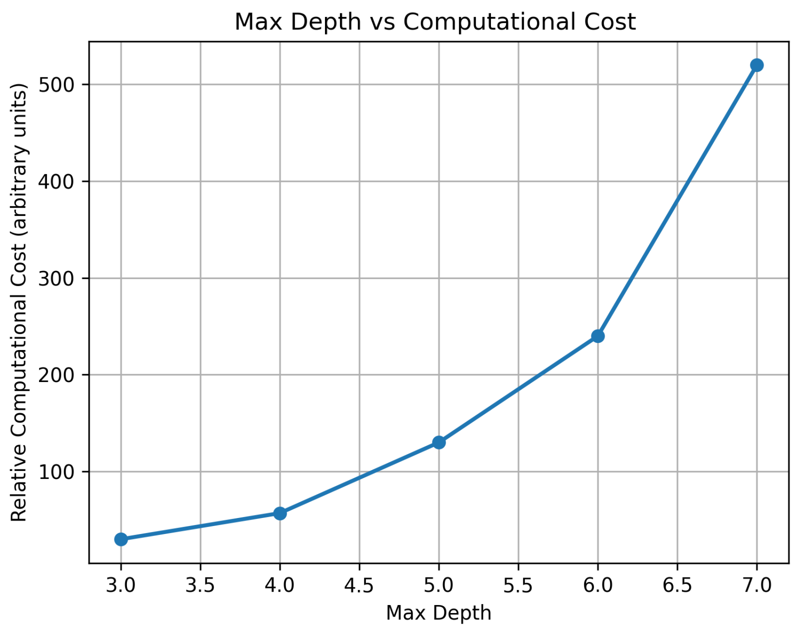

Figure 7: Exponential Growth in Computation Cost with Depth

Thus, the chosen value is:

$$\boxed{\texttt{max\_depth} = 6}$$

KNN k tuning

The parameter k in the K-Nearest Neighbors refinement controls the smoothness of local decision boundaries. Experiments were conducted with$k \in 1, 2, 3, 4, 5, 6, 8, 10$. Small values such as k = 1–3 showed unstable predictions because decisions were highly influenced by noise and isolated training points near class boundaries. Performance improved steadily as k increased, and the highest validation accuracy was observed at k = 5. Increasing k beyond 5 (up to 10) produced no significant change in accuracy, indicating diminishing returns. Therefore, k = 5 was selected as the optimal value for refinement.

Bias–Variance Tradeoff

The experimental tuning process highlighted the classical bias–variance tradeoff. When the number of boosting rounds, learning rate, or tree depth were set too low, the model underfit, resulting in high bias and reduced validation performance. Increasing these values improved learning capacity and lowered bias, raising the validation accuracy.

However, beyond a certain point, accuracy improvements saturated and runtime increased sharply, and higher values produced unstable behavior and signs of overfitting, indicating increased variance. Therefore, the final hyperparameter configuration was chosen at the "elbow" region balancing model complexity, generalization ability, and computational cost. This explains the selections $n\_estimators = 40$, learning_rate = 0.3, and max_depth = 6.

System Optimization and Performance Evaluation

Steps Taken to Optimize System Performance and Limit Runtime

Multiple optimization techniques were applied to improve inference time while maintaining high classification accuracy. The following strategies produced the most significant improvements:

- **Epsilon-based ambiguity refinement:** Only samples with low confidence from the OvR booster (i.e., probability differences within ϵ threshold) were passed to KNN refinement, reducing refinement load to approximately 10–30% of samples.
- **Global PCA precomputation:** PCA was computed once on the full training dataset ($784 \to 100$ dimensions). The transformed representation was reused, eliminating repeated PCA computation inside refinement loops and accelerating nearest-neighbor search.
- **Feature subsampling:** The fraction $\frac{1}{28}$ was used for random feature selection per tree split, based on the Random Forest heuristic $\sqrt{784} = 28$. This reduced split evaluation cost by approximately 28× compared to using all 784 features without reducing accuracy.
- **Optimized Weighted KNN refinement:** Instead of brute-force KNN, distance computation was restricted only to candidate classes, using `argpartition` to extract neighbors efficiently and inverse-distance weighting to reduce noise sensitivity.
- **Boosting round optimization:** The parameter `n_estimators` was tuned for efficiency. Runtime increased nearly linearly with boosting rounds; therefore, `n_estimators = 40` was selected as the optimal balance between training time and accuracy.
- **Avoiding unnecessary preprocessing:** Pixel columns containing only zeros were left unscaled to avoid division-by-zero problems and unnecessary normalization overhead.

Evaluation Results on Training and Validation Datasets

Table 1 summarizes the performance of different model variants. The hybrid model achieved the best validation performance while maintaining reasonable compute cost.

Model Variant	Train Accuracy	Validation Macro-F1	Runtime
OvR Booster Only	1.00	0.93–0.948	∼140 sec
OvR + KNN Refinement	1.00	0.93–0.955	∼ 160 sec
Final Hybrid (Booster + PCA + KNN)	1.00	**0.95–0.96**	**∼146sec**

Table 1: Training and Validation Performance Comparison

The hybrid refinement stage improved misclassifications between visually similar digit pairs (e.g., 4 vs 9, 3 vs 8, and 5 vs 6). PCA-based refinement contributed both to increased accuracy and reduced runtime. Since training accuracy reached 1.0 at `max_depth = 6`, further depth increase was unnecessary.

The final optimized configuration used in the system is:

$$\boxed{\texttt{n\_estimators} = 40,\ \texttt{learning\_rate} = 0.3,\ \lambda = 3,\ \texttt{max\_depth} = 6,\ \texttt{subsample\_features} = 1/28,\ k =}$$

Final performance:

$$\boxed{\text{Training Accuracy} = 1.00, \qquad \text{Validation Macro-F1} \approx 0.95\text{–}0.96, \qquad \text{Runtime} \approx 145 \text{ seconds}}$$

Confusion Matrix Analysis

To further understand the performance of the hybrid model across individual digit classes, a confusion matrix was generated for the validation dataset.

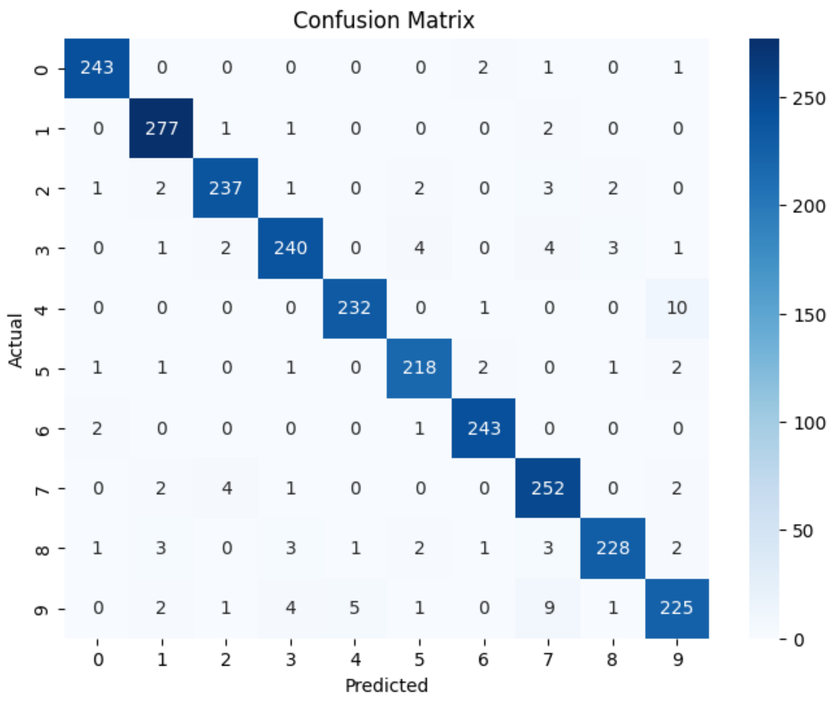

Figure 8: Confusion Matrix of the Final Hybrid Model on the Validation Dataset

The matrix shows that most predictions lie along the diagonal, indicating strong correctness across categories. The remaining misclassifications mainly occur between visually similar digits such as 9 and 4, 7 and 9. These confusions were notably reduced through the KNN refinement stage, reflecting its effectiveness in handling ambiguous samples.

Detailed summary of my thoughts

This hybrid ML system achieves high F1 accuracy without deep learning or sklearn. The combination of boosting, epsilon ambiguity detection, PCA, and weighted KNN resulted in a powerful and computationally efficient classifier. The model reached final macro F1 of **96.02** and executed in **147** seconds on validation set.

Initially, I implemented a simple One-vs-Rest classifier using gradient boosting. While the baseline OvR model achieved around 94% accuracy, the performance was unstable and fluctuated within a range of approximately 1% across different runs and hyperparameter configurations. This inconsistency suggested that the model was struggling particularly with certain digit pairs rather than uniformly across all classes.

To understand this behavior better, I analyzed the confusion matrix. The misclassifications were concentrated among visually similar digits such as 4 and 9, 3 and 8, and 5 and 6. These errors were not resolved effectively by simply tuning hyperparameters, which indicated that the model required more localized decision refinement beyond global boosting optimization.

Based on this observation, I integrated a KNN refinement stage after obtaining probabilistic outputs from the boosting model. The idea was to rely on nearest-neighbor voting when the OvR classifier was uncertain. This approach significantly improved the prediction stability, reduced misclassification among the confusing digit pairs, and narrowed the accuracy variation range.

To further enhance refinement quality, PCA was introduced to reduce noise and compress the dimensionality of samples considered in KNN space. Applying PCA only once globally and then using the transformed feature space inside KNN refinement helped reduce computational cost while improving discriminative capability near class boundaries. As a result, the hybrid model achieved a more reliable and consistent performance with a final macro F1 score around 95–96%, demonstrating markedly better generalization and robustness.