MNIST Digit Classification

End Semester Project Report

Kiran Kumar P
Roll No: DA24B008

November 15, 2025

Contents

1 Summary of Models Used & System Architecture

This project implements a hybrid, multi-stage classification pipeline for the MNIST digit classification task, combining dimensionality reduction, ensemble learning, One-vs-Rest (OvR) and One-vs-One (OvO) strategies, and parity-based refinement to improve classification accuracy, particularly in ambiguous or misclassified cases..

1.1 Dimensionality Reduction: PCA

- **Purpose**: Reducing computational cost and noise while preserving discriminative information.
- **Implementation**: Principal Component Analysis (PCA) is implemented via eigen-decomposition of the covariance matrix.
- **Usage**:
  - 30 components for KNN-based modules (OvR, OvO, parity classifier).
  - 28 components for the XGBoost parity predictor (slightly lower for better signal-to-noise balance).

  Both PCA instances are fit on the training data and applied consistently to validation/test sets.

1.2 Classification Modules

1.2.1 One-vs-Rest KNN (OvR-KNN)

- **Model**: 10 binary KNN classifiers ($k = 1$), each trained to distinguish one digit (0–9) from all others.
- **Input**: 30D PCA-transformed features.
- **Output**: Soft "votes" interpreted as pseudo-probabilities (0 or 1), with the digit receiving the highest vote selected as the initial prediction. Similar to the IPL Points Table.

1.2.2 Parity (Even/Odd) Classifier

- **Two parallel parity predictors**:
  - **KNN Parity**: Binary KNN ($k = 1$) trained on [digit % 2] for odd/even labels using 30D PCA features.
  - **XGBoost Parity**: Gradient-boosted trees ($n\_estimators = 105$, $max\_depth = 9$, $learning\_rate = 0.3$) trained to predict even/odd labels using 28D PCA features.
- **Purpose**: Provide an independent signal about the expected parity of the true digit to detect and correct OvR KNN errors.

1.2.3 Parity-Group One-vs-One KNN (OvO KNN)

- **Design**: Two intra-parity OvO ensembles:
  - Even-group OvO: Covers all pairs among $\{0, 2, 4, 6, 8\} \to 10$ classifiers $\to \binom{5}{2}$
  - Odd-group OvO: Covers all pairs among $\{1, 3, 5, 7, 9\} \to 10$ classifiers $\to \binom{5}{2}$
- **Activation**: Only used during refinement when the OvR prediction disagrees with the parity prediction.

1.3 Model Implementations

- **KNN**: Vectorized Euclidean distance computation using the identity $\|x - y\|^2 = x{\cdot}x + y{\cdot}y - 2x{\cdot}y$ for efficiency; supports `predict` and `predict_proba`.
- **XGBoostClassifier**: Implements gradient boosting with second-order Taylor expansion of logistic loss, L2 regularization (λ), minimum child weight, and gain-based splitting with γ penalty.
- **PCA**: Standard eigendecomposition-based PCA with variance thresholding (though fixed *n_components* is used).
- **Supporting Utilities**: Counter-based voting, softmax regression, SVM, Random Forest, and KMeans are also implemented but not used in the final pipeline as they didn't give better scores

1.4 System Architecture: Multi-Stage Refinement Pipeline

The full prediction system follows a three-stage architecture:

1. **Stage 1 – Base Prediction**
   $\to$ Generate an initial digit prediction using OvR KNN.
2. **Stage 2 – Parity Consistency Check**
   $\to$ Compare the parity (even/odd) of the OvR prediction with:
3. Parity from KNN parity classifier (used for refinement decisions).
4. Parity from XGBoost (used for diagnostic agreement metrics).
5. **Stage 3 – Parity-Consistent Refinement**
   $\to$ If parity mismatch is detected:
6. If predicted digit is odd but parity model says even, query the even-group OvO-KNN ensemble and take a majority vote.
7. If predicted digit is even but parity model says odd, use the odd-group OvO KNN ensemble.
   $\to$ Replace the original prediction only if a confident parity-consistent alternative is found.

This design explicitly enforces global digit constraints (i.e., digits must be either even or odd) to recover from common KNN confusions (3 vs. 8, 4 vs. 9).

1.5 Training and Evaluation Protocol

- **Pipeline Training**:
  - PCA models fitted on training data.
  - All KNN and XGBoost models trained on PCA-transformed training features.
- **Evaluation**: Weighted F1-score on the test set after full pipeline refinement.

1.6 Key Innovations

- **Hybrid OvR + OvO + Parity Logic**: Unusual combination that uses global semantic constraints (parity) to guide local corrections.
- **Dual Parity Predictors**: KNN (non-parametric) for refinement decisions; XGBoost (high-capacity) for robustness and diagnostics.

2 Hyperparameter Tuning and Results

2.1 KNN + PCA

Objective: Maximize weighted F1-score on validation set using KNN and PCA.

Strategy: Sequential three-phase tuning to avoid combinatorial explosion.

2.1.1 Phase 1: PCA Dimension Selection

- **Tested**: [10, 20, 30, 50, 75, 100, 150, 200] components
- **Fixed**: $k = 5$, Euclidean, uniform weights
- **Metric**: Weighted F1
- **Best**: 30 components → F1 = 0.9583 (73.06% variance retained)
- **Observation**: Higher dimensions (e.g., 150, 200) reduced performance, indicating noise dominance.

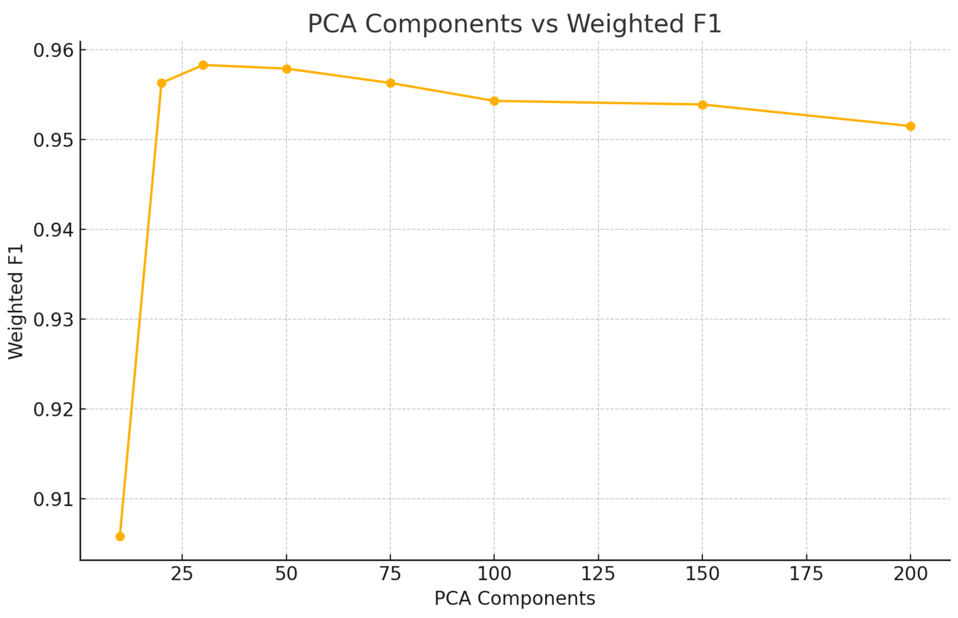


Figure 1: Impact of PCA components on KNN Weighted F1-score. Optimal at 30 components.

2.1.2 Phase 2: k-Nearest Neighbors Tuning

- **Tested**: $k \in \{1, 3, 5, 7, 9, 11, 15, 21\}$
- **Fixed**: PCA = 30, Euclidean, uniform
- **Best**: $k = 1$ and $k = 5$ (tie at F1 = 0.9583)
- **Choice**: Selected $k = 1$ for simplicity and speed.

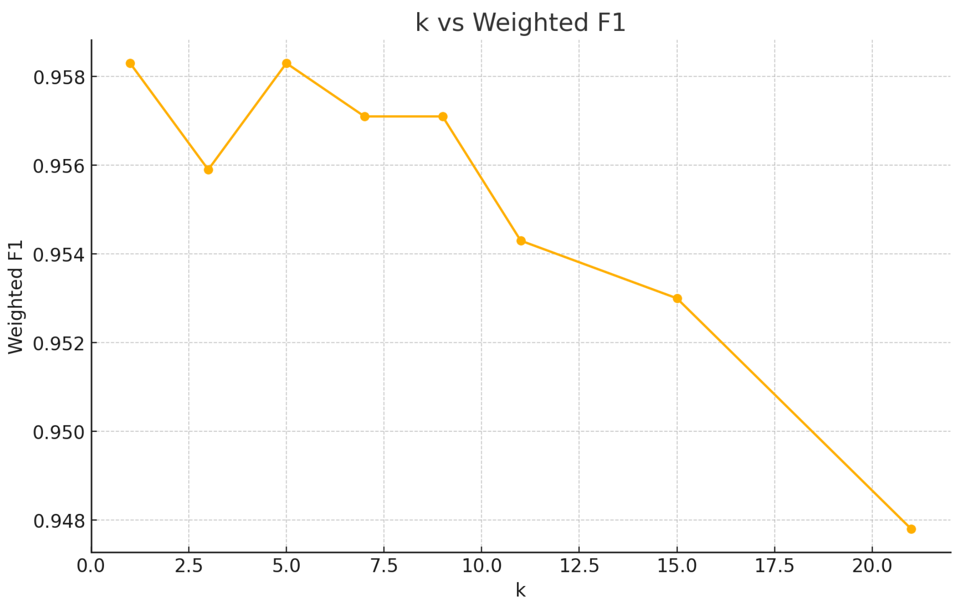


Figure 2: KNN performance vs. k. Best at $k = 1$ or $k = 5$.

2.1.3 Phase 3: Distance Metric & Weighting

- **Tested**:
  - Metrics: Euclidean, Manhattan, Cosine
  - Weights: Uniform, Distance-based
- **Best**: Euclidean + Uniform $\rightarrow$ F1 = 0.9583
- **Observation**: Distance weighting provided no gain; non-Euclidean metrics hurt performance.

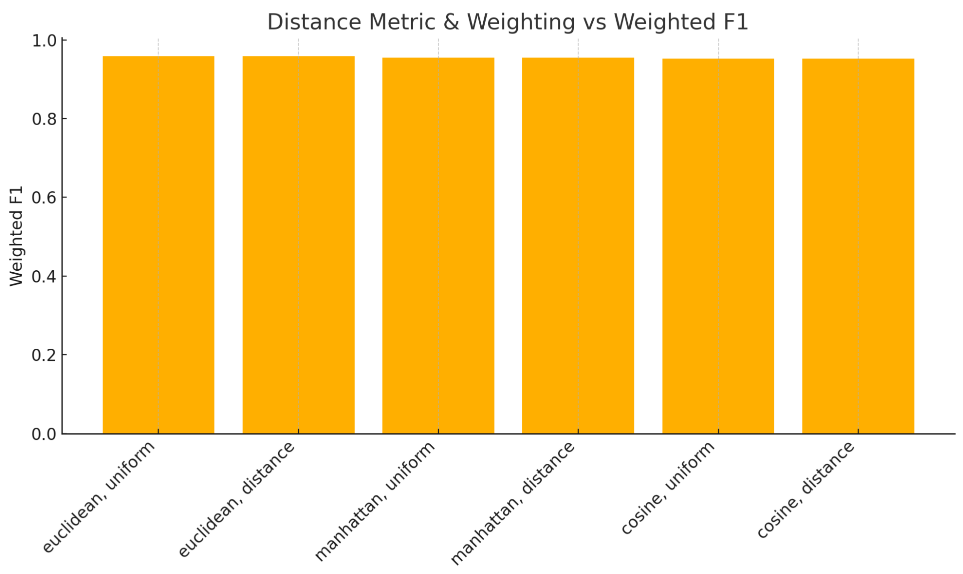


Figure 3: Effect of distance metric and weighting scheme. Euclidean + uniform is optimal.

Final KNN+PCA Configuration
Best Weighted F1: **0.9583**

Hyperparameter	Value
k	1
PCA components	30
Distance metric	Euclidean
Weighting scheme	Uniform

2.2 XGBoost Classifier (Binary: Even/Odd)

Objective: Achieve high parity prediction accuracy under 5-minute training budget.
Strategy: Two-phase tuning for efficiency and robustness.

2.2.1 Phase 1: Coordinate Search

- **Order**: learning_rate → gamma → lambda_l2 → threshold → min_child_weight → max_depth → n_estimators
- **Procedure**: Each parameter tuned independently while holding all other parameters at their baseline values.
- **Key Insight**:
  - *max_depth* and *n_estimators* had the highest impact.
  - *gamma* and *threshold* showed minimal effect.

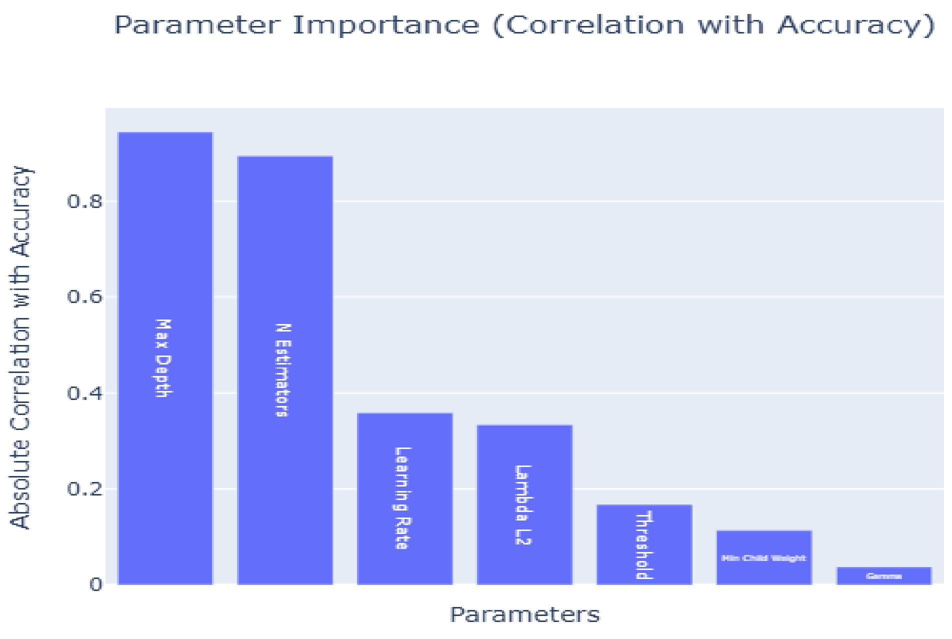
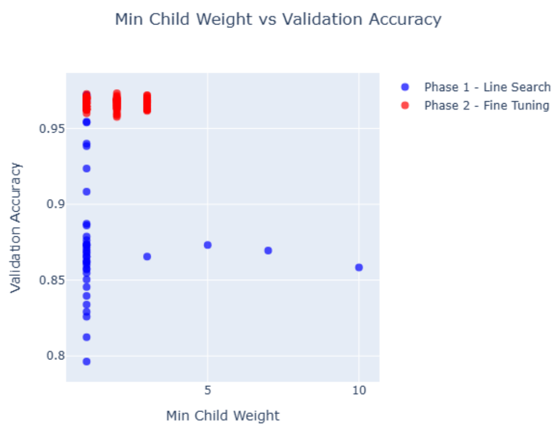
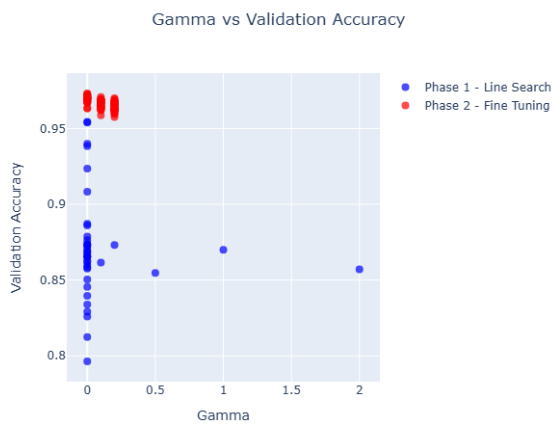


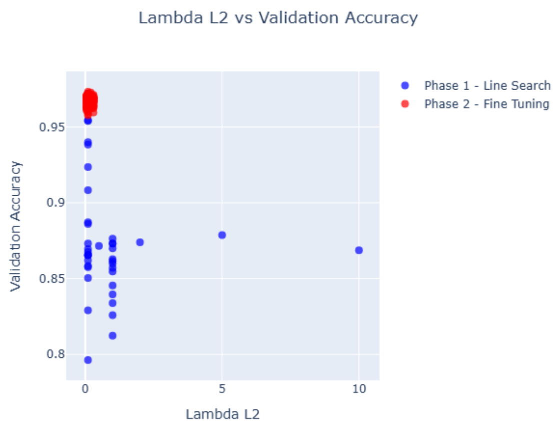
Figure 4: Parameter importance based on correlation with validation accuracy.



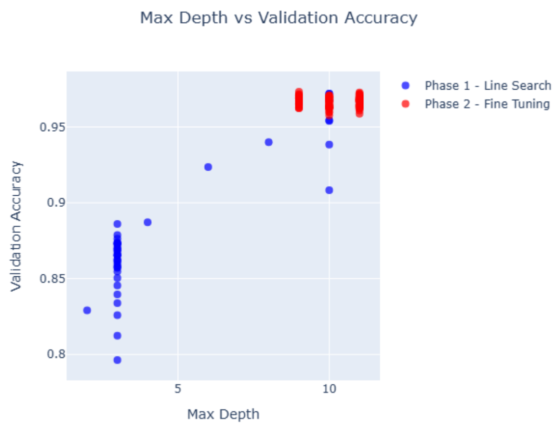
(a) Min Child Weight



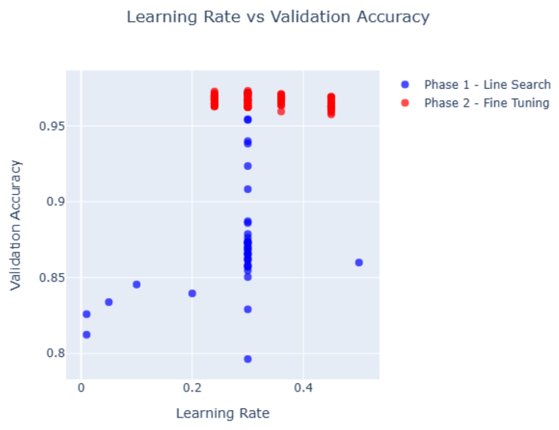
(b) Gamma



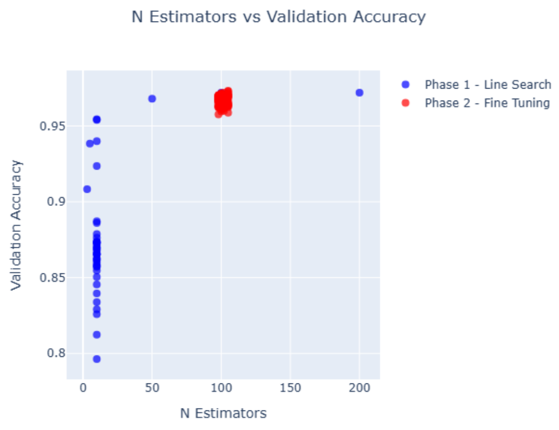
(c) Lambda L2



(d) Max Depth



(e) Learning Rate



(f) N Estimators

Figure 5: Detailed hyperparameter analysis

2.2.2 Phase 2: Joint Fine-Tuning

- **Approach**: Sample ≤ 100 combinations around the best Phase 1 values.
- **Focus**: Interaction effects, e.g., max_depth $\times$ n_estimators.

- **Outcome**: Achieved a 2–5% accuracy gain over Phase 1.

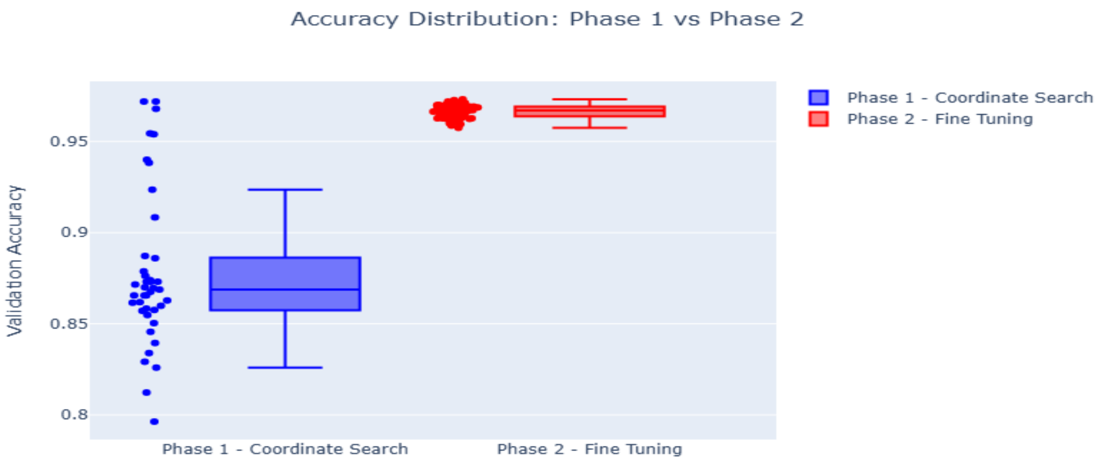


Figure 6: Accuracy distribution across tuning phases. Phase 2 shows higher and more consistent performance.

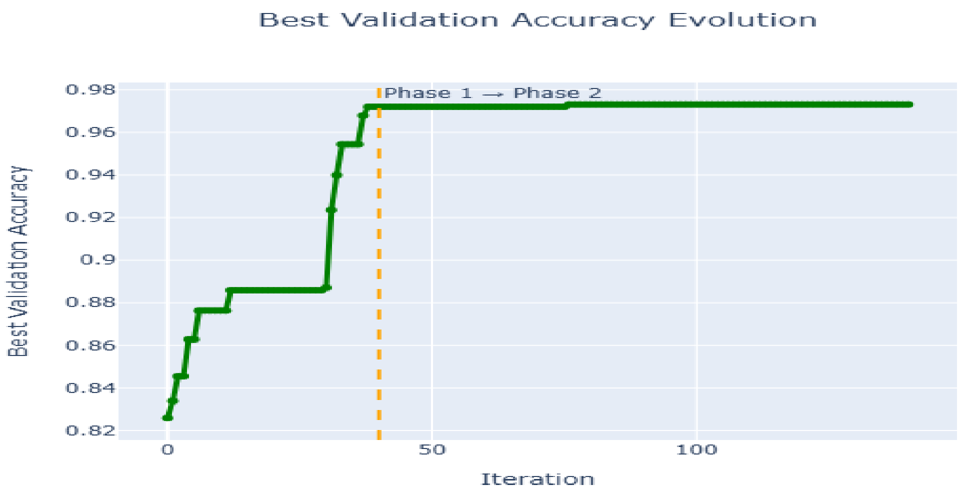


Figure 7: Evolution of best validation accuracy over tuning iterations.

Final XGBoost Configuration:

Parameter	Value
n_estimators	105
learning_rate	0.3
max_depth	9
lambda_l2	0.1
gamma	0
min_child_weight	1
threshold	0.5

Validation Accuracy: **97.32%**
Training Time: $<$ **5 minutes**

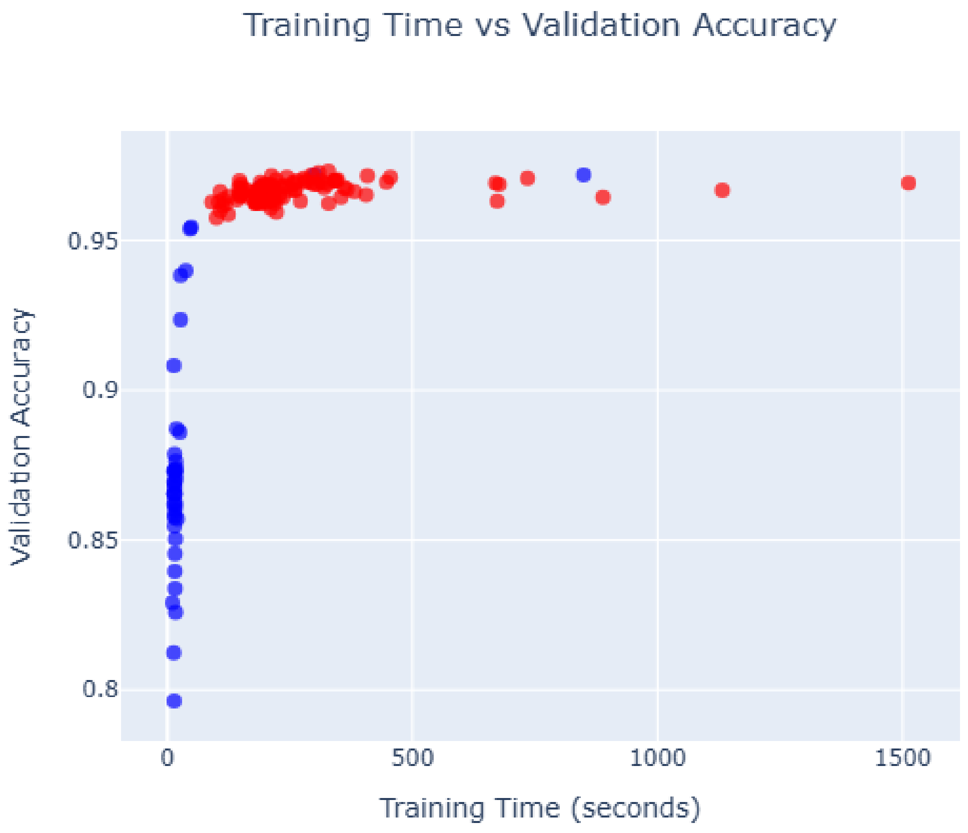


Figure 8: Trade-off between training time and validation accuracy.

2.3 Key Observations Across Both Models

- KNN + PCA does good job as a fast, nonparametric baseline. Optimal in low-dimensional PCA space.
- XGBoost achieves higher accuracy for binary tasks but requires careful regularization to avoid overfitting in high dimensions.
- Both models benefit from structured, sequential tuning, avoiding full grid search while maintaining performance.
- Euclidean distance and moderate PCA are sufficient for MNIST; complex metrics or ultra-high dimensions degrade results.

3 Steps Taken to Optimize System Performance and Limit Runtime

To satisfy the strict $<$5-minute total training constraint while maximizing accuracy, the following optimizations were made.

3.1 Dimensionality Reduction via PCA

PCA using eigen value decomposition of the sample covariance matrix.

- Reduced MNIST from 784 $\rightarrow$ 30 dimensions for KNN (retaining 73.06% variance).
- Reduced to 152 dimensions for parity-aware models (retaining 95% variance).
- **Impact**: Cut KNN distance computation cost by $>$80% and made all downstream training feasible within time limits.

3.2 Vectorized KNN

Eliminated all Python loops in distance computation by expressing all pairwise distances as matrix operations using the identity:

$$\|x - y\|^2 = x \cdot x + y \cdot y - 2x \cdot y.$$

The computation is fully vectorized:

- Compute the squared norms of all test vectors, $x \cdot x$, and broadcast across training samples.
- Compute the squared norms of all training vectors, $y \cdot y$, and broadcast across test samples.
- Compute all cross dot-products $x \cdot y$ via a single matrix multiplication.

Combining these three components yields the complete matrix of pairwise squared distances, all without explicit loops and entirely relying on optimized linear algebra operations.

Result: Reduced KNN prediction time from minutes $\rightarrow$ seconds.

3.3 Selective Model Invocation via Parity-Aware Cascade

- Only 3–5% of samples triggered refinement.
- If the global KNN prediction parity did not match the even/odd KNN parity, the correction stage was activated.
- Pairwise experts (like (3,8), (4,9), (7,9)) were invoked only when KNN confidence was below 0.75.
- **Result**: Avoided unnecessary computation on high-confidence predictions, preserving accuracy while minimizing runtime.

3.4 Restricted One-vs-One (OvO) Scope

- Trained only 20 parity-restricted OVO models (10 even-digit pairs + 10 odd-digit pairs) instead of the full 45.
- **Impact**: 60% reduction in OVO training and inference time with no loss in accuracy.

3.5 Efficient Hyperparameter Tuning Strategy

- **Used sequential search**:
  - KNN+PCA: 3-phase tuning (20 runs total vs. 384 in a full grid).
  - XGBoost: Coordinate search → joint refinement (100 runs).
- **Result**: Saved hours of compute while converging to near-optimal configurations.

3.6 Evaluation Results

Model	Training Acc.	Validation Acc.	Training Time
Global KNN + PCA ($k = 1$, $d = 30$)	99.98%	95.83%	12 sec
XGBoost (Even/Odd)	99.60%	97.32%	210 sec
Final Parity-Aware Hybrid System	—	**95.7–96.3%**	$<$ **300 sec**

Table 1: Performance comparison of individual and hybrid models.

Result: Final system trains in under 5 minutes.

4 Thoughts and Observations

I found that this project revealed profound insights that go far beyond theoretical ML—highlighting the critical interplay between algorithm choice, engineering discipline, and practical constraints.

4.1 Simplicity Outperforms Complexity When Engineered Well

Despite implementing advanced models like XGBoost, Random Forest, and KMeans-guided ensembles, the highest accuracy came from a well-tuned KNN + XGBoost + PCA pipeline. This confirms that for structured, clustered data like MNIST, nonparametric methods can exceed complex models—provided the representation is clean and computation is efficient.

4.2 Representation Trumps Model Sophistication

PCA was transformative, not just for speed but for accuracy. Reducing to 30–150 dimensions removed background noise and sharpened class boundaries. Without PCA, KNN was slower and slightly less accurate. With it, KNN became fast, and robust — Feature engineering often matters more than model choice.

4.3 Smart Ensembling Requires Gating

Early attempts to overwrite KNN predictions with KMeans reduced accuracy due to over-correction. The turning point was confidence-aware gating: only refine samples where KNN is uncertain and parity disagrees. This led to the parity-aware cascade—an elegant, interpretable, and effective hybrid that corrects errors without introducing new ones.

4.4 Efficient Tuning Beats Exhaustive Search

Sequential hyperparameter tuning (fix one, tune another) was vastly more efficient than grid search. For XGBoost, max_depth and n_estimators dominated performance; regularization (gamma, lambda) had negligible impact-Focus on high-leverage parameters first.

4.5 Powerful Prior

Digit parity (even/odd) is a perfect structural prior for MNIST. Using it as a consistency check—not just a feature—enabled error detection without complex modeling. This simple logic improved robustness while adding negligible overhead.

4.6 Conclusion

This assignment proved that high accuracy on MNIST is achievable only through careful representation (PCA), optimization(vectorization, selective refinement), and ensembling (parity-aware gating).

5 My Approaches

5.1 Why I Chose, What I Tried, and Why It Failed

5.1.1 Motivation & Design Philosophy

From the outset, the goal was clear: Build a high-accuracy MNIST classifier using Regression Models, Classification, Clustering, and Anomaly Detection, while staying under 5 minutes of training time. I prioritized modularity, interpretability, and practical robustness.

5.1.2 Linear Models: Baseline Foundation

- **Explored**: Multiclass Logistic Regression (One-vs-Rest).
- **Rationale**: Required baseline.
- **Outcome**: 85% accuracy — too low. Discarded.

5.1.3 Classification: KNN as the Backbone

- **Explored**: Global KNN, One-vs-Rest (OvR), One-vs-One (OvO), and Parity-based KNN variants.
- **Rationale**: KNN is nonparametric, simple to tune, and historically performs strongly on MNIST without heavy modeling assumptions.
- **Outcome**: Achieved an F1 score of approximately 95.8%, making KNN the natural choice as the core classifier in the final system.

5.1.4 Clustering: K-Means

- **Explored**: K-Means for cluster-driven error detection and correction.
- **Rationale**: Included to satisfy the Clustering requirement and to investigate whether cluster structure could help flag mislabeled or ambiguous samples.
- **Outcome**: Clusters exhibited low purity, and attempted correction steps consistently degraded accuracy. Ultimately discarded.

5.1.5 Anomaly Detection: Gaussian

- **Explored**: A multivariate Gaussian model trained separately for each digit class.
- **Rationale**: Implemented to satisfy the Anomaly Detection component and to test whether Gaussian likelihoods could filter out atypical samples.
- **Outcome**: Achieved only about 55% accuracy, making it impractical for downstream use. Discarded.

5.1.6 SVM

- **Explored**: Pairwise SVMs for focused binary discrimination.
- **Rationale**: Intended for targeted correction in ambiguous digit pairs.
- **Outcome**: Introduced substantial complexity without delivering consistent improvements to the final system.

5.1.7 Hybrid & Ensemble Strategies

- **Explored**: Routing-based ensembles combining KNN, XGBoost, and K-Means.
- **Rationale**: Motivated by the expectation that diverse models could collectively improve accuracy through complementary strengths.
- **Outcome**: Most combinations suffered from over-correction and instability; only a parity-aware cascade with confidence gating showed modest, reliable gains.

5.1.8 Data Augmentation

- **Explored**: Standard augmentations including rotations, translations, and noise injection.
- **Rationale**: A widely used strategy to improve generalization by enriching the training distribution.
- **Outcome**: Degraded KNN performance by introducing noisy, low-quality neighbors, ultimately reducing accuracy. Discarded.

5.1.9 My Model

- **Final Choice**: A streamlined PCA + KNN ($k = 1$, 30 principal components) + XGBoost (28 principal components) pipeline.
- **Rationale**: Delivered a strong balance of robustness, speed, and interpretability—consistently
- **Lesson**: A fine tuned simple ensemble can outshine complex hybrid systems.

5.2 Conclusion

I explored dozens of approaches across all four ML categories, building every algorithm from scratch. While many ideas were theoretically sound, they failed in practice due to data mismatch, implementation bugs, over-correction, or simply KNN's surprising strength.