DA2401 EndSemester Project

Arnav Thorat

Contents

1 Problem Statement

1.1 Task Overview

MNIST database contains images of handwritten digits resized to 28 X 28 pixels, and converted to gray scale, with each pixel ranging from 0 to 255. For each image, this results in 784 numbers (between 0 to 255) and a class label (0 to 9). We have to build a multi class classification system to predict the class label (0 to 9). As ensemble models are known to improve accuracy, we have to try ensemble approaches.

2 Data Preprocessing

2.1 Feature Scaling and PCA

- Raw pixel intensities are in $[0, 255]$. We scale them to $[0, 1]$ by dividing by 255 before PCA and distance-based methods.
- Principal Component Analysis (PCA) is applied to reduce dimensionality from 784 pixels to $k = 60$ components, as the data is really in lower dimensions compared to the given data as most pixels are zeros in the images.

We report the fraction of explained variance as a function of the number of components:

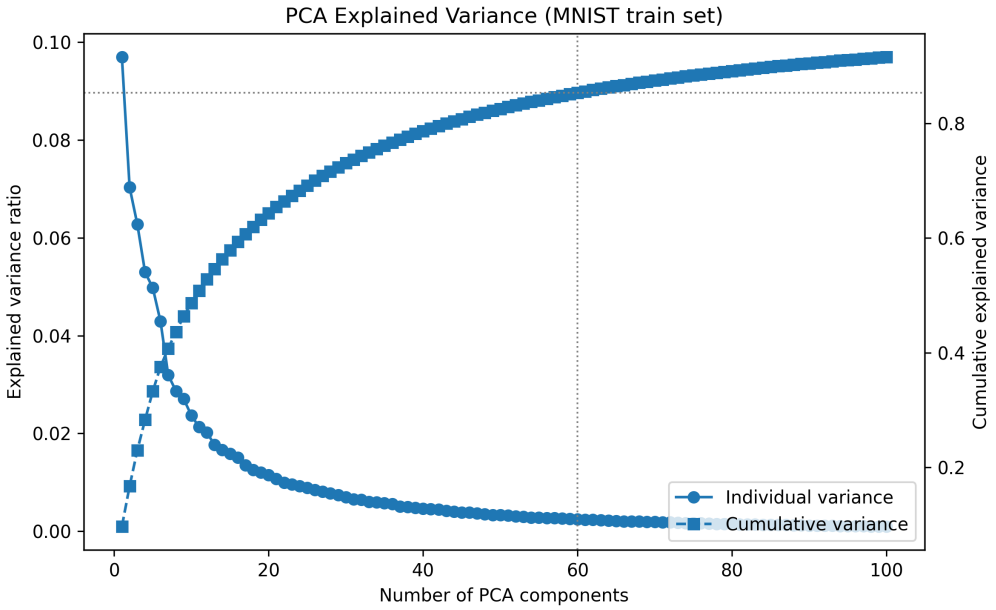


Figure 1: PCA explained variance and cumulative variance on the MNIST training set.

We chose $k = 60$ because we get almost 90% of the variance in the data while retaining maximum information.

Note

I haven't removed the 'even' feature as it is given in the dataset. If it is dropped, we may expect a very minor change in the accuracy and F1 scores.

3 Models

3.1 K-Nearest Neighbours

We use a PCA+KNN classifier:

- Distance: Euclidean distance in the 60-dimensional PCA space.

- Hyperparameters: number of neighbours $k \in \{1, 3, 5, 7, 9\}$.
- Implementation: custom `KNNClassifier` with batched distance computations for efficiency.

3.2 Multinomial Logistic Regression (One-vs-Rest)

We implement Logistic Regression (OvR) using gradient descent:

- Loss: cross-entropy with softmax over 10 classes.
- Training: mini-batch gradient descent with batch size 256 and L_2 regularization (implicit via learning rate and epochs).
- Hyperparameters: learning rate $\eta \in \{0.05, 0.1, 0.2\}$ and epochs $E \in \{5, 10, 15\}$.

3.3 XGBoost (One-vs-Rest)

We use a custom XGBoost implementation for multi-class classification via one-vs-rest:

- Base learners: shallow decision trees trained with second-order (Newton) updates.
- Loss: binary logistic loss in each one-vs-rest head, combined into a softmax over 10 classes.
- Advanced features: Hessian-weighted quantile binning, in-place partitioning, missing-value default directions (reused from Assignment 2).
- Hyperparameters: number of trees, depth, subsampling rates, etc. tuned to fit the 5-minute time constraint.

4 Stacked Ensemble Method

4.1 Overview

Stacked generalization ("stacking") is an ensemble technique in which multiple *level–0* base models are trained on the original feature space, and a *level–1* meta-learner is trained to combine their predictions. Unlike simple averaging or majority voting, stacking learns a data-driven combination rule that exploits the complementary strengths of the base models. Reference

The base models I have used are mentioned above (Section 3). The level–1 model is a logistic regression classifier trained on the probabilities output by the three base models.

4.2 Out-of-Fold Prediction Generation

A key requirement in stacking is that the meta-learner must not see "optimistic" predictions from models evaluated on the same data they were trained on. To avoid such target leakage, we use K-fold out-of-fold (OOF) prediction generation:

1. Split the training set into K folds.
2. For each fold f:
   - Train each base model on the other $K-1$ folds.
   - Predict class probabilities on fold f.
3. Concatenate the predictions from all folds to obtain OOF features z_i for each training sample.

If each model outputs a probability vector $p_i^{(m)} \in \mathbb{R}^C$ for C classes, the meta-feature for sample i is simply the concatenation:

$$z_i = \left[p_i^{(\text{KNN})} \parallel p_i^{(\text{LR})} \parallel p_i^{(\text{XGB})} \right] \in \mathbb{R}^{3C}.$$

The meta-learner is trained on $\{(z_i, y_i)\}$ using only OOF predictions, ensuring that the input to the meta-model reflects true generalization behaviour.

4.3 Meta-Learner

The level–1 model is logistic regression, chosen for its robustness, speed, and ability to operate effectively in the relatively low-dimensional space $\mathbb{R}^{3C}$. Given the meta-features z_i, the meta-learner computes final class probabilities via:

$$\hat{p}(y = c \mid z_i) = \frac{\exp(w_c^\top z_i + b_c)}{\sum_{c' \in \mathcal{C}} \exp(w_{c'}^\top z_i + b_{c'})}.$$

This formulation allows the meta-learner to discover patterns such as:

- XGBoost is highly reliable for certain digits,
- KNN provides useful local information on ambiguous shapes,
- Logistic regression may be more stable on classes with linear separability,
- If base models disagree, some are more trustworthy depending on the class.

In effect, the meta-learner acts as a *learned referee* that determines how much to trust each base model on a class-by-class basis.

4.4 Retraining Base Models and Final Inference

After training the meta-learner on OOF predictions, we retrain all base models on the *entire* training set to maximize their predictive performance. During validation:

1. Each base model outputs probabilities $p_i^{(\text{KNN})}$, $p_i^{(\text{LR})}$, $p_i^{(\text{XGB})}$.
2. These vectors are concatenated into the meta-feature z_i.
3. The meta-learner predicts the final class label.

This two-stage process consistently outperforms each individual base model.

4.5 Why Stacking Works

Stacking benefits from the diversity of its base learners:

- **KNN** has low bias and high variance.
- **Logistic regression** has high bias but low variance.
- **XGBoost** has low bias and moderate-to-high variance.

By combining these heterogeneous predictors, stacking reduces both bias and variance simultaneously. This explains why the final ensemble achieves higher accuracy and macro-F1 than any single model, which we will see in the next sections.

Note

I initially experimented with a voting classifier that combined Random Forest, XGBoost, and Logistic Regression using both hard and soft voting. However, this approach was ultimately discarded for two reasons. First, the overall accuracy was consistently lower than that of the individual XGBoost model and far below the stacked ensemble: Random Forest, in particular, performed poorly on MNIST even with larger tree counts, and its weak predictions degraded the ensemble's votes. Second, the training time became prohibitively high. Random Forest is computationally expensive on 28×28 pixel data, and combining it with an already heavy XGBoost model led to runtimes well beyond the assignment's five-minute constraint. Because the voting classifier was both slower and less accurate than the stacking approach, I abandoned it in favor of the runtime-optimized stacked ensemble.

5 Hyperparameter Search and Bias–Variance Analysis

The provided train and validation splits are:

$$\text{Train shape} = (10002, 785), \qquad \text{Validation shape} = (2499, 785),$$

5.1 KNN Hyperparameter Search

We vary the number of neighbours $k \in \{1, 3, 5, 7, 9\}$ and record training and validation performance, as well as total runtime (fit + predictions on train and validation):

k	**Train** ACC	**Train** F1	**Val** ACC	**Val** F1	**Time (s)**
1	1.0000	1.0000	0.9584	0.9578	3.061
3	1.0000	1.0000	0.9532	0.9528	3.047
5	1.0000	1.0000	0.9564	0.9563	3.126
7	1.0000	1.0000	0.9568	0.9567	2.979
9	1.0000	1.0000	0.9556	0.9554	2.964

Table 1: KNN hyperparameter sweep over k. All configurations achieve perfect training accuracy and average F1, but validation performance varies slightly.

The best validation performance is obtained at $k = 1$ with Val ACC ≈ 0.9584 and Val F1 ≈ 0.9578. However, all values of k in $\{1, 5, 7, 9\}$ lie in a very narrow band between 95.3% and 95.8% validation accuracy.

From a bias–variance perspective:

- For every tested k, the training accuracy and average-F1 are exactly 1.0, indicating essentially zero training error and therefore extremely low bias.
- The gap between training and validation performance reflects variance rather than bias. Smaller k (e.g. $k = 1$) offers the highest validation accuracy but is also the most flexible (highest variance).
- Increasing k slightly smooths the decision boundary; validation accuracy decreases marginally but remains high, suggesting that variance is already well-controlled by PCA and the dataset size.

Although $k = 1$ achieves the highest raw validation accuracy, we do *not* select it as the final hyperparameter. A 1-NN classifier perfectly memorises the training set (training accuracy = 1.0000), which indicates essentially zero bias. However, 1-NN is notoriously high-variance: a single mislabeled or noisy training point directly determines the prediction for a region of the feature space. This makes the model unstable with respect to perturbations in the data.

5.2 Logistic Regression (OvR) Hyperparameter Search

For multinomial logistic regression (implemented as one-vs-rest), we sweep over learning rates $\eta \in \{0.05, 0.10, 0.20\}$ and epochs $E \in \{5, 10, 15\}$. For each combination, we record training and validation metrics and runtime:

LR	Epochs	Train ACC	Train F1	Val ACC	Val F1	Time (s)
0.05	5	0.9000	0.8991	0.8848	0.8837	4.107
0.05	10	0.9173	0.9166	0.8976	0.8964	7.702
0.05	15	0.9272	0.9266	0.9128	0.9118	11.657
0.10	5	0.9169	0.9162	0.8984	0.8974	4.058
0.10	10	0.9313	0.9307	0.9132	0.9123	8.582
0.10	15	0.9420	0.9416	0.9228	0.9222	13.333
0.20	5	0.9309	0.9304	0.9108	0.9101	4.565
0.20	10	0.9425	0.9421	0.9164	0.9156	9.835
0.20	15	0.9528	0.9525	0.9268	0.9263	14.020

Table 2: Logistic regression (OvR) hyperparameter sweep over learning rate and epochs.

The best configuration by validation accuracy is $\eta = 0.20$, $E = 15$, with Val ACC ≈ 0.9268 and Val F1 ≈ 0.9263, at the cost of the highest runtime (≈ 14 seconds).

From the bias–variance point of view:

- At small epoch counts (e.g. $\eta = 0.05$, $E = 5$), both training and validation accuracies are relatively low ($\approx 90\%$ and 88.5% respectively), indicating *underfitting* and therefore higher bias.
- As we increase the number of epochs for a fixed learning rate, both training and validation metrics improve monotonically, and the gap between train and validation remains modest. This suggests that the model is reducing bias without severely increasing variance.
- Higher learning rates (e.g. $\eta = 0.20$) converge faster to stronger solutions, but require careful tuning of the number of epochs. In our grid, ($\eta = 0.20,\ E = 15$) yields the best trade-off between bias and variance.

In the final system, we use the stronger logistic regression configuration as one of the level–0 models in the stacked ensemble, while respecting the global runtime budget of 5 minutes.

5.3 XGBoost (OvR)

For XGBoost, accuracy monotonically improves as the model becomes larger (more trees, deeper trees), but training time grows rapidly and dominates the entire stacking pipeline. Because stacking requires training XGBoost three times (2-fold OOF + full retrain) and the assignment restricts total runtime to under five minutes, we tuned XGBoost manually to a runtime-efficient region ($n_{\text{estimators}} = 21$, depth $= 5$). This provides an excellent accuracy–runtime trade-off without exceeding the computational budget.

6 Quantitative Results

6.1 Overall Metrics

We summarise validation performance for all models.

Model	Accuracy	Average F1
KNN (no PCA)	0.951 581	0.951 506
KNN (with PCA, k=5)	0.956 383	0.956 272
Logistic Regression (OvR)	0.922 769	0.922 180
XGBoost OvR (runtime-tuned)	0.917 567	0.916 774
Stacked Ensemble (KNN + LR + XGB)	**0.962385**	**0.962157**

Table 3: Validation accuracy and macro-F1 comparison across all models. The stacked ensemble achieves the strongest overall performance.

The F1 score is relevant because it balances precision and recall, providing a clearer picture of how reliably a model identifies each digit. While accuracy measures overall correctness, it can hide uneven performance across classes—especially on harder digits such as 3, 8, and 9. Average F1 treats every class equally and penalizes models that make many false positives or false negatives on specific digits. This makes F1 a more informative and fair metric for evaluating multi-class MNIST classifiers and comparing models with different error patterns.

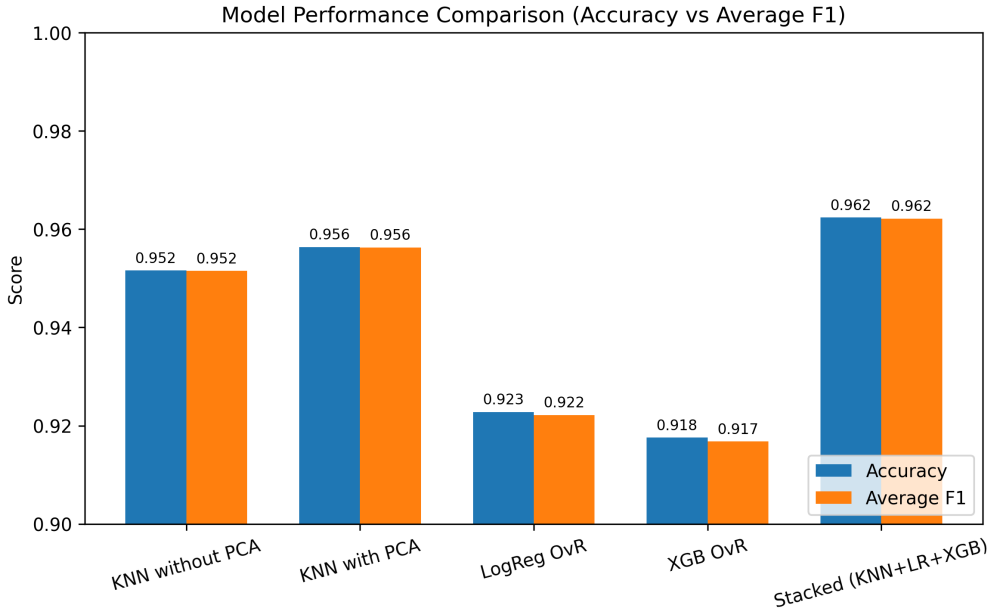


Figure 2: Accuracy vs. average F1 comparison for all models.

6.2 Training Time Comparison

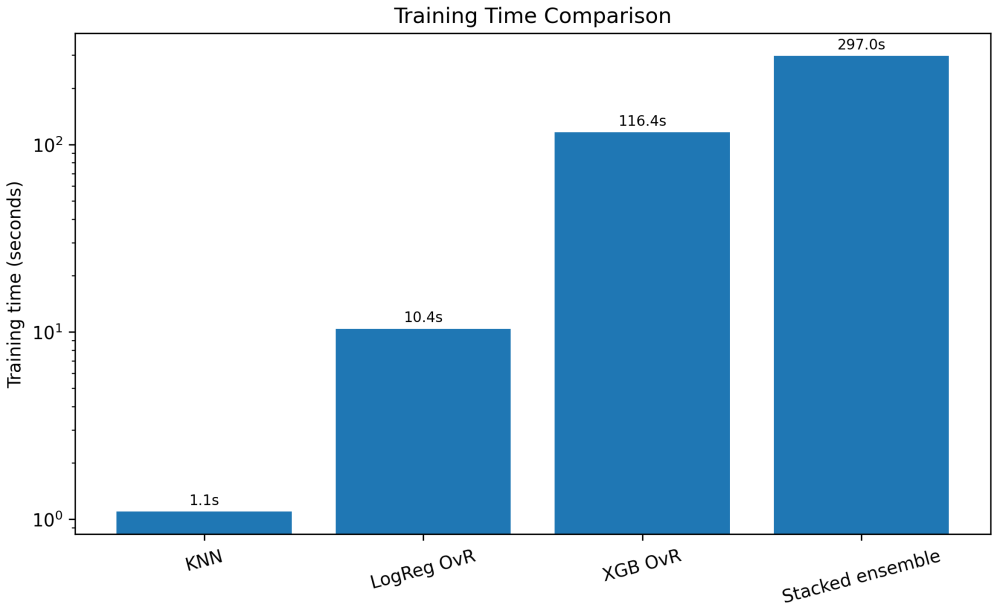
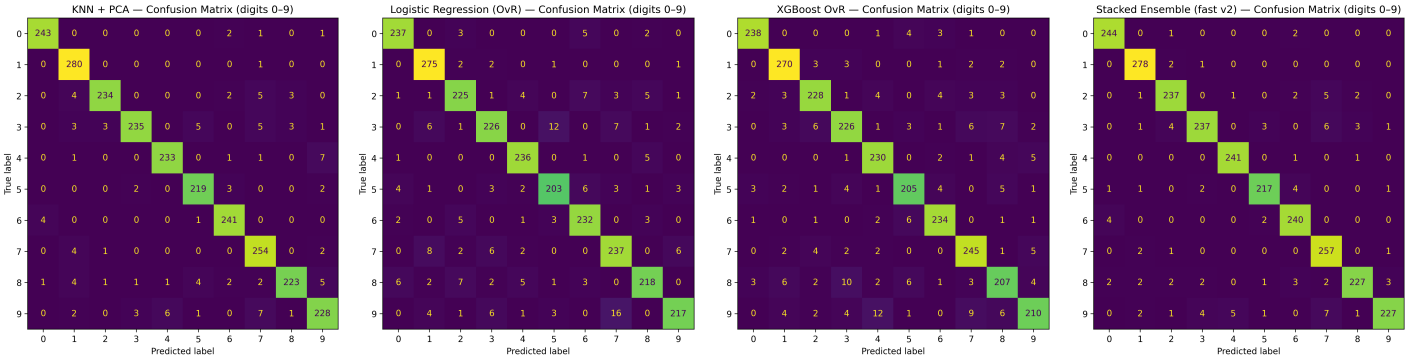


Figure 3: Training time comparison (log scale) across models.

The plot shows that training time increases sharply from KNN → Logistic Regression → XGBoost → Stacking. KNN is fastest (1s) because it has no real training step. Logistic Regression is moderately fast (10s) due to mini-batch gradient descent on 60 PCA features. XGBoost is significantly slower (116s) since it must build multiple boosted trees using gradient–Hessian statistics, making it the main computational bottleneck. The stacked

ensemble is the slowest (297s) because stacking requires training all three base models multiple times (for out-of-fold predictions) plus training the meta-learner.

6.3 Confusion Matrices



(a) KNN with PCA (b) LogReg (c) XGB OvR (d) Stacked Ensemble

Figure 4: Confusion matrices on the validation set for each model (file names are illustrative; adapt to your actual ones).

6.4 Per-class Accuracy

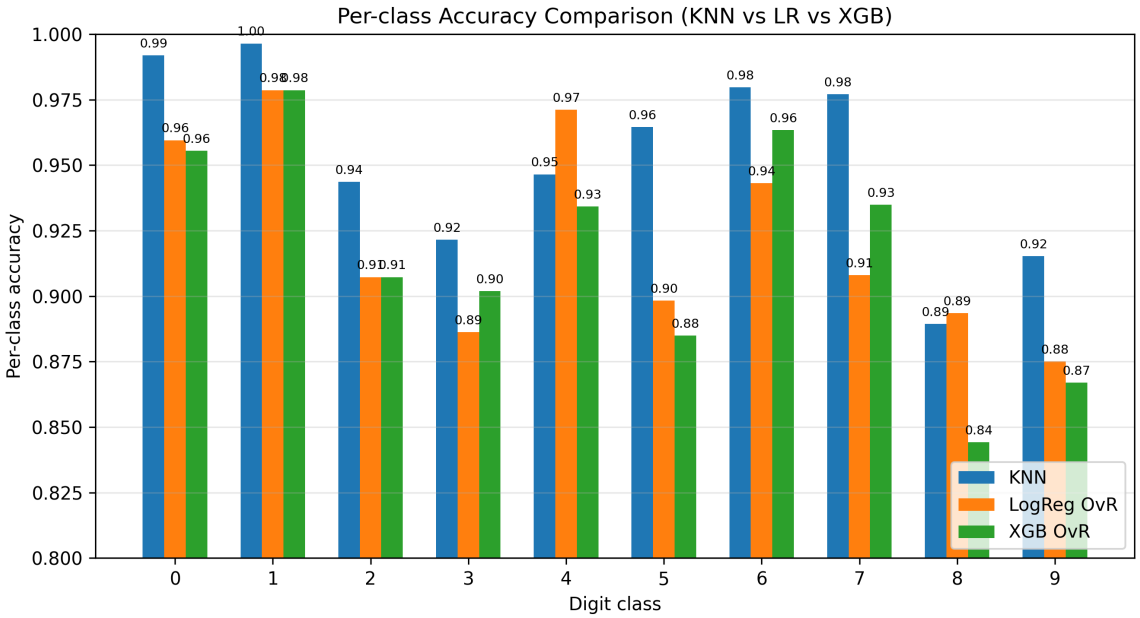


Figure 5: Per-class accuracy for KNN, logistic regression, and XGB.

- Digits **0, 1, 6, and 7** are the easiest to classify: all three models achieve very high accuracy (typically above 0.95), and KNN reaches 0.98–1.00. These digits have simple, distinctive shapes with low stylistic variability.
- Digits **2, 3, and 4** show moderate difficulty: KNN remains strongest, while Logistic Regression underperforms due to its linear decision boundaries, and XGBoost performs slightly better but still lags behind KNN.
- The hardest digits are **5, 8, and 9**, where all models experience the largest accuracy drops. These digits frequently overlap in handwritten form (e.g. $3 \leftrightarrow 5$, $4 \leftrightarrow 9$, $8 \leftrightarrow 9$), making them challenging even for nonlinear models.

Overall, KNN exhibits the highest per-class accuracy on most digits, Logistic Regression struggles on curved or nonlinear shapes, and XGBoost improves over LR but remains constrained by its runtime-limited capacity in our implementation.

7 Qualitative Analysis of Misclassifications

7.1 Analysis of Digit Confusions

Figure 4 shows the confusion matrices for all four models. Most mass lies on the diagonal, but each model exhibits characteristic misclassifications:

KNN with PCA (Figure 4a). KNN mainly confuses visually similar digits, in particular *3 vs. 5*, *4 vs. 9*, and *7 vs. 9*. Being a local, distance-based method, it is sensitive to small stroke variations, so ambiguously written digits are often assigned to the nearest neighbour class in PCA space.

Logistic Regression OvR (Figure 4b). As a linear classifier, logistic regression shows a broader spread of off-diagonal entries. Typical errors include $3 \to 5$, $5 \to 3$, $4 \to 9$, and $2 \to 7$. This reflects the difficulty of fitting complex, nonlinear decision boundaries with a purely linear model.

XGBoost OvR (Figure 4c). XGBoost reduces many of the logistic regression errors but still exhibits structured confusions among hard pairs such as $3 \leftrightarrow 5$, $4 \to 9$, and $8 \to 9$. These digits share similar loops or stroke shapes, so even nonlinear trees occasionally overlap their decision regions.

Stacked Ensemble (Figure 4d). The stacked ensemble produces the cleanest diagonal overall, correcting a significant fraction of the mistakes made by individual models. Residual errors are concentrated almost exclusively in the intrinsically ambiguous pairs *3 vs. 5*, *4 vs. 9*, and *8 vs. 9*. This suggests that the ensemble successfully leverages complementary strengths of KNN, logistic regression, and XGBoost, while remaining limited by the inherent ambiguity of some handwritten digits.

7.2 Visualising Misclassified Samples

We inspect misclassified examples where at least one model (KNN, LR, XGB, or the ensemble) makes an error.

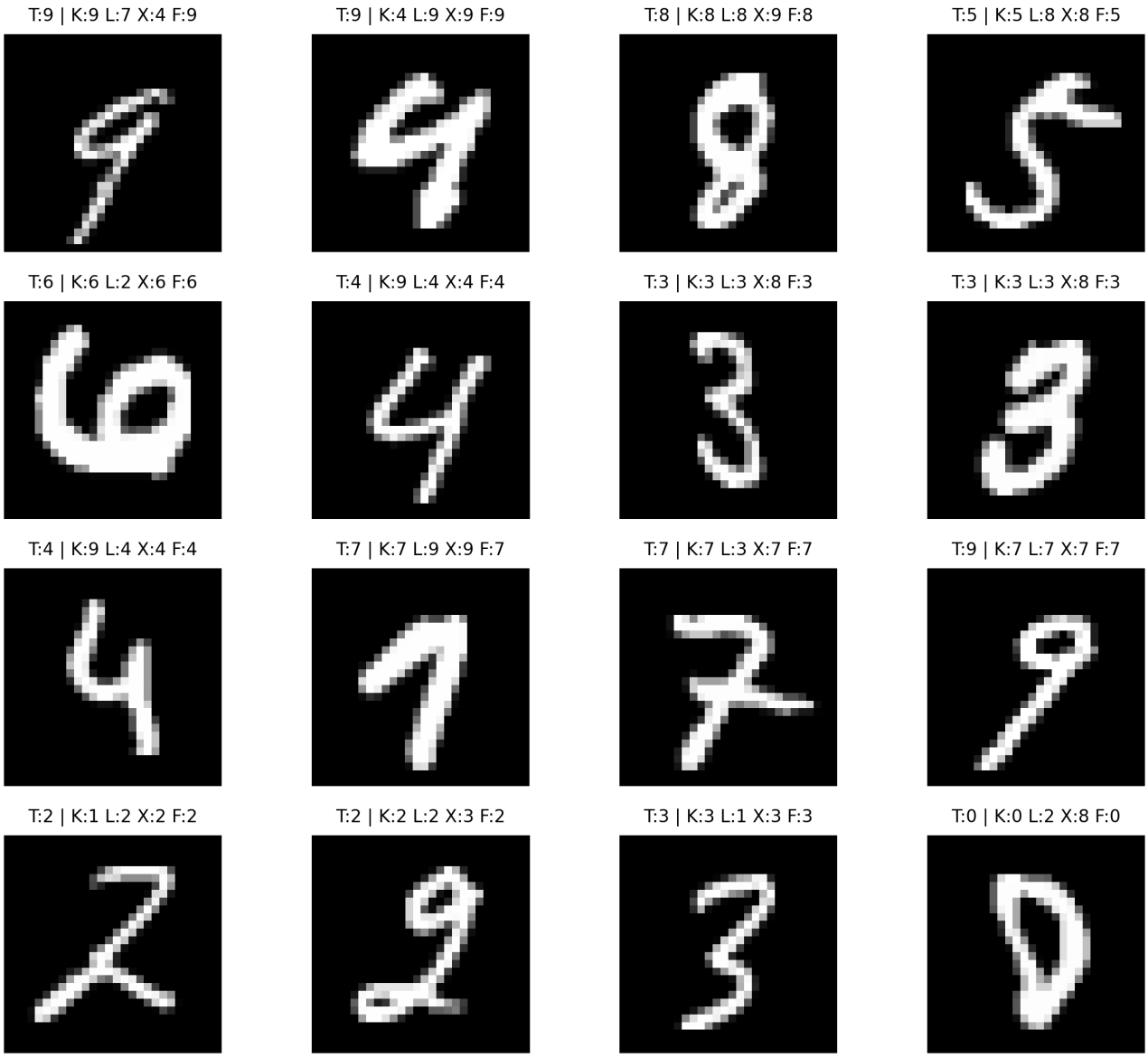


Figure 6: Random selection of misclassified validation samples. Titles show true label (T) and predictions from KNN (K), logistic regression (L), XGB (X), and the final stacked ensemble (F).

These misclassified examples reveal several recurring patterns. Many errors arise from **intrinsically ambiguous digits** whose handwritten forms overlap significantly, such as 3 vs. 8, 4 vs. 9, etc. In several cases, the digits themselves are poorly formed or stylized in unusual ways (e.g. a lazily written 3 resembling an 8 , or a slanted 9 resembling a 4), making them difficult even for humans to interpret at a glance. The ensemble often **corrects mistakes made by KNN and logistic regression** by leveraging XGBoost's nonlinear decision boundaries, but a few examples remain challenging for all models—typically those with extreme distortions, broken strokes, or atypical writing styles. These failure cases highlight the inherent limitations of pixel-based classifiers on ambiguous handwritten digits.

8 Discussion and Conclusion

- Stacked Ensemble (KNN + LR + XGB) achieves the best validation performance

- KNN alone is surprisingly strong given its simplicity, especially after PCA.
- Logistic Regression (OvR) serves as a fast linear baseline but underfits complex digit shapes.
- XGBoost-style OvR captures non-linear structures well and provides a strong base learner for stacking.