# DA2401 - Endsem Report

Krish Yadav- DA24B043

15th November

## 1. Introduction

For this lab-endsem-project, I have explored multiple machine learning models for classifying handwritten digits (MNIST). At the start I had no clear idea which model would balance both accuracy and runtime, so I started with some simple baselines and gradually moved towards more complex models. My entire workflow throughout was kind off an iterative loop, try something, analyze the behaviour, tune a bit, observe errors, and move forward.

My main goal was to achieve a good validation accuracy while keeping the total runtime below the allowed limit i.e. 5-6 minutes.

This report summarizes:

1. All models that I used and overall architecture
2. Hyperparameter tuning summary for each model
3. Steps taken to optimize performance and stay under runtime limits
4. My thoughts and observations throughout this whole process

# 2. Summary of Models Used & System Architecture

## Preprocessing

The MNIST CSV files were normalized to the range $[0, 1]$ by dividing each pixel by 255. I did no extra augmentation or preprocessing because the goal was more of an algorithmic comparison instead of squeezing maximum dataset performance.

## Models Implemented

All models were implemented from scratch inside `algorithms.py`. The following architectures were used:

1. **One-vs-Rest Logistic Regression**
   Basic binary logistic regression repeated for each digit (0 to 9). This served as my first proper baseline because I had already implemented logistic for binary classifcation my the assignment 2.

2. **Softmax Regression (Full Batch)**
   Multiclass logistic regression with cross-entropy loss.

3. **Mini-Batch Softmax Regression**
   Same as softmax regression but trained in mini-batches to speed up convergence.

   But these models (1 -3) still try to learn linear decision boundaries so i had to move on to non-linear model next.

4. **Decision Tree** (for Random Forest)

5. **Random Forest Classifier**
   Bagging + random feature selection. I experimented with many combinations of number of trees, depth, etc.
   Still Random forest doesnt learn from its previous mistakes i.e. it build trees independently, so i moved onto the XGB models.

6. **XGBoost (Binary)** This I had implemented in the assignment 2 of lab, so I later it wrapped using One-vs-Rest for multiclass classification.

7. **XGBoost Multiclass (Softmax Version)**
   This version trains $K$ trees per boosting round (one per class). So also this ended up being the final chosen model for the endsem.

8. **KNN**
   This model I added purely out of curiosity to see how a simple distance–based non-parametric method performs on MNIST. Since KNN does no training and just stores the dataset, I wanted to test whether a very simple naive method can still give me reasonable accuracy. To my surprise it gave a decent performance.

9. **Ensemble Classifier** (Softmax + RF + XGB-MC)
   A weighted probability averaging ensemble. I only used this in testing because even though it is fun to experiment, the runtime became too high for final submission usage.

### 2.3 System Architecture

The architecture was simple:

1. Load train and validation CSV
2. Preprocess (normalize)
3. Train selected model(s)
4. Measure accuracy, weighted F1 and runtime
5. Compare and pick best configuration

All testing scripts were kept separate from the actual `main.py` used for final submission.

## 3. Hyperparameter Tuning Summary

So here is the link to a single pdf containing all my efforts for hyper paramater tuning. CLICK ON THIS

# 4. Ensemble Attempts

I also explored whether combining multiple models would help push the accuracy even further. The idea was simple: use a weighted probability averaging ensemble of three different types of learners:

$$\text{Ensemble} = \{\text{Mini-Batch Softmax},\ \text{Random Forest},\ \text{XGBoost Multiclass}\}$$

Each model contributes class probabilities, and the ensemble takes a weighted average:

$$P(y = k|x) = \frac{1}{\sum w_i} \sum_i w_i \, P_i(y = k|x)$$

Weights used:

$$w = [1,\ 1,\ 2]$$

since XGBoost-MC was the strongest individual performer.

## 4.1 What Models and Settings Were Tried?

I tried three different ensemble configurations:

- **Run 1 - Full-strength models**
  - Mini-Batch Softmax (50 epochs)
  - Random Forest: 25 trees, depth 20
  - XGBoost Multiclass: 25 estimators

  **Outcome:** Could not finish within the allowed time budget — RF ($\sim 250s$) + XGB ($\sim 330s$) alone crossed 600+ seconds. So i discarded this immediatley.

- **Run 2 - Medium-sized ensemble (best feasible run)**
  - Mini-Batch Softmax (2.19 sec)
  - Random Forest: 15 trees, depth 10 (118.19 sec)
  - XGBoost Multiclass: 15 estimators, depth 3 (189.51 sec)

  **Total Time:** 309.89 sec **Accuracy:** 95.68% **Weighted F1:** 0.9568 This run completed successfully, but was still slower and less accurate than XGBoost Multiclass alone.

- **Run 3 - Light ensemble (just for testing)**
  - Random Forest: 15 trees
  - XGBoost Multiclass: 10 estimators

  **Outcome:** $\sim$ 95.5% accuracy, finishing around 250 seconds. Still did not outperform the single best model.

## 4.2 Summary of All Ensemble Experiments

| Run | RF Config | XGB Config | Time (sec) | Accuracy |
|---|---|---|---|---|
| Run 1 (Full) | 25 trees, depth 20 | 25 estimators | $> 600$ | – |
| Run 2 (Best) | 15 trees, depth 10 | 15 estimators | 309.89 | 95.68% |
| Run 3 (Light) | 15 trees | 10 estimators | $\sim 250$ | $\sim 95.5\%$ |

## 4.3 Why Ensemble Was NOT Used in the Final System

After completing all runs, I decided not to use the ensemble in my final submission. The reasons were very clear:

- The ensemble **never outperformed** the standalone XGBoost Multiclass model (which reached around 96.4–96.7%).
- It significantly **increased the runtime**, often nearing or exceeding the limit.
- Averaging predictions diluted the strong XGBoost model with weaker models.
- XGBoost Multiclass **already acts like an ensemble internally** (multiple trees per class per round), so extra ensembling added no real benefit.

In short, although the ensemble was fun to experiment with, it did not improve accuracy and made the system unnecessarily slow. So the final system uses only the **XGBoost Multiclass** model.

# 5. Final Selected Model: XGBoost Multiclass

## 5.1 Final Performance Table

| Model | Accuracy | Weighted F1 | Runtime (sec) |
|---|---|---|---|
| OvR Logistic Regression | 89.11% | 0.8903 | 13.5 |
| Softmax Regression (MB) | 92–93% | $\sim 0.92$ | 2–3 |
| Random Forest | 93.47% | 0.9344 | 283 |
| OvR XGBoost | 95.71% | 0.9571 | 392 |
| **XGBoost Multiclass (Final)** | **96.44%** | **0.9644** | **357** |
| KNN (k=5) | 95.24% | – | 18 |
| Ensemble | 95.68% | 0.9568 | 309 |

The final model chosen had the best tradeoff between runtime and accuracy.

## 5.2 Confusion Matrix (Validation Set)

To analyse the final model behaviour in more detail, I computed the confusion matrix for the XGBoost Multiclass classifier on the 2,499–sample validation set. This helps understand which digits the model tends to confuse the most and whether there are any systematic patterns.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 241 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 2 | 0 |
| 1 | 0 | 275 | 0 | 3 | 0 | 0 | 0 | 2 | 0 | 1 |
| 2 | 1 | 0 | 240 | 0 | 3 | 0 | 1 | 0 | 3 | 0 |
| 3 | 0 | 1 | 0 | 246 | 0 | 3 | 0 | 4 | 0 | 1 |
| 4 | 2 | 0 | 0 | 0 | 238 | 0 | 0 | 0 | 3 | 0 |
| 5 | 0 | 2 | 0 | 3 | 0 | 216 | 0 | 2 | 0 | 3 |
| 6 | 3 | 0 | 0 | 0 | 0 | 0 | 238 | 0 | 5 | 0 |
| 7 | 0 | 1 | 0 | 2 | 0 | 0 | 0 | 251 | 0 | 7 |
| 8 | 2 | 0 | 3 | 0 | 3 | 0 | 2 | 0 | 234 | 0 |
| 9 | 0 | 2 | 0 | 5 | 0 | 1 | 0 | 8 | 0 | 232 |

Table 1: Confusion Matrix for XGBoost Multiclass on the Validation Set.

# 6. Thoughts & Observations

This entire assignment was like a mini research cycle. A few key observations:

- **Tree-based models consistently beat linear models** on MNIST, even with simple depth control.
- **Random Forest is also decent**, but becomes slow as trees and max features increase.
- **XGBoost (OvR) is extremely accurate**, but training 10 separate binary models is computationally very expensive.
- **XGBoost Multiclass gave the most balanced results**. It captures nonlinear boundaries effectively while staying under the runtime limit(5-6 min)
- **Mini-batch softmax is the fastest**, but because of linear decision boundaries performance is limited.
- **KNN turned out surprisingly strong**, reaching around 95% with almost zero training cost.
- **Ensemble was interesting but unnecessary**. The extra time did not justify the tiny improvement and sometimes hurt accuracy.

Overall, this project helped me understand the tradeoff between model expressiveness and computational budget. A well-tuned XGB multiclass model clearly stood out as the most practical choice.