Report: MNIST Classification Pipeline

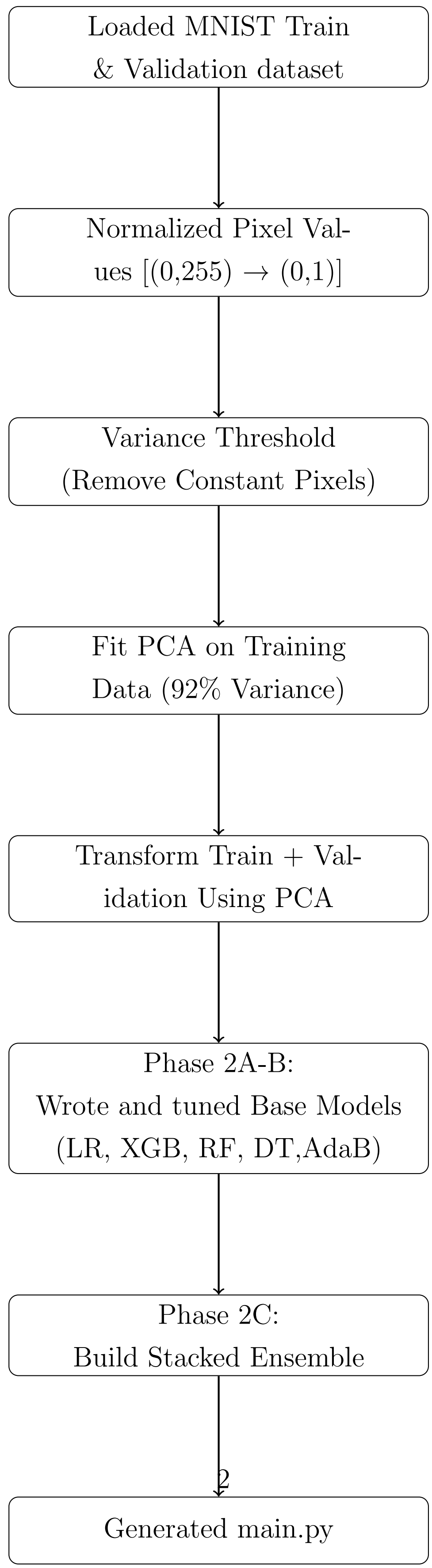
Abhijeet Kumar
DA24B001

This report details my process for building, tuning, debugging, and finalizing a machine learning pipeline from scratch in NumPy to classify the given MNIST dataset.

Note:

Sir and TA's, After doing this project, i found few learning outcomes that i have mentioned in the last section with heading **"My Thoughts and Learnings"**. please verify them and let me know whether i am right or not.
Thanks!

My Pipeline Flowchart



Summary of Models Used & System Architecture

My project is built as a sequential pipeline, starting from raw data and ending with a final, optimized, and stacked model. I wrote all my core algorithms from scratch using only NumPy.

System Architecture

My pipeline is broken into several phases:

1. **Phase 1: Preprocessing & Feature Engineering**
   - I start by loading the `MNIST_train.csv` and `MNIST_validation.csv` data.
   - I normalize all pixel values from the [0, 255] range to a [0, 1] scale.
   - I'm applying a `VarianceThreshold` ($1e{-5}$) to remove constant, all-black pixels that provide no information.
   - Crucially, I am using **Principal Component Analysis (PCA)** to reduce dimensionality. I am fitting the PCA only on the training data (to prevent data leakage) and transforming both the train and validation sets.

PCA Dimensionality Reduction Diagram

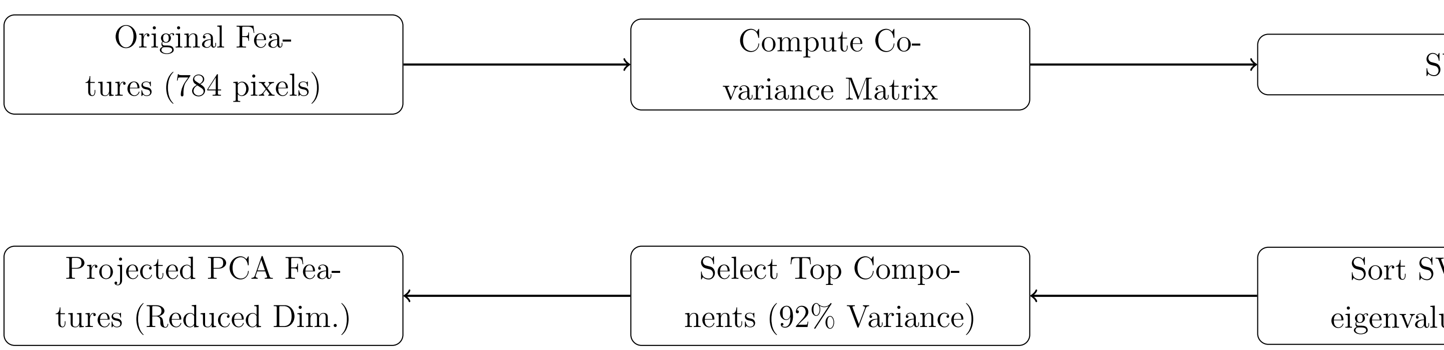


Figure 2: PCA: Transforming 784 Raw Pixels Into Lower-Dimensional Features

Phase 2A-B: Base Model Tuning

In this phase, I'm individually tuning all my models. My goal is to find the best-performing model for each algorithm (based on Weighted F1) that can train in under 300 seconds.

Phase 2C: Stacked Ensemble

- I'm selecting my best-performing and most diverse models from the tuning phase to act as "base learners."
- I am then training a "meta-model" (a `MultinomialLogisticRegression`) on the predictions from these base models.
- A key step here is that I am **One-Hot Encoding** the predictions before feeding them to the meta-model.This drastically increased my F1 score.
- I have stacked **Softmax + XgBoost** becuase of their nature of solving the problem, like,
  if test data will be linear my sofmax will work or if complex my xgboost will handle.

Summary of Hyper-parameter Tuning and Results

My tuning phase was a process of discovery, identifying which models were suited for my PCA-transformed data and which were not.

Model	**Best F1-Weighted**	**Best Hyperparameters**	**Key Observation**
MulticlassXGBoost	**0.9129**	`n_est=60, max_depth=5, lr=0.1`	**Best Model.** Found non-linear patterns even after PCA.
Logistic Regression	**0.9031**	`n_epochs=100, lr=0.1, batch=64`	**Best Linear Model**. Extremely fast and highly accurate.
Random Forest	0.8067	`n_estimators=70, max_depth=14`	**Rejected.** Heavy overfitting and very slow (764s).
Decision Tree	0.7778	`max_depth=14, min_samples=5`	**Rejected.** Overfit (95% train accuracy).

XGBoostSimple	∼0.65	(Various)	**Rejected.** Too weak; stumps cannot learn MNIST.
AdaBoost	∼0.65	(Various)	**Rejected.** Weak and extremely slow (1200+ seconds).

Stacking Architecture Diagram

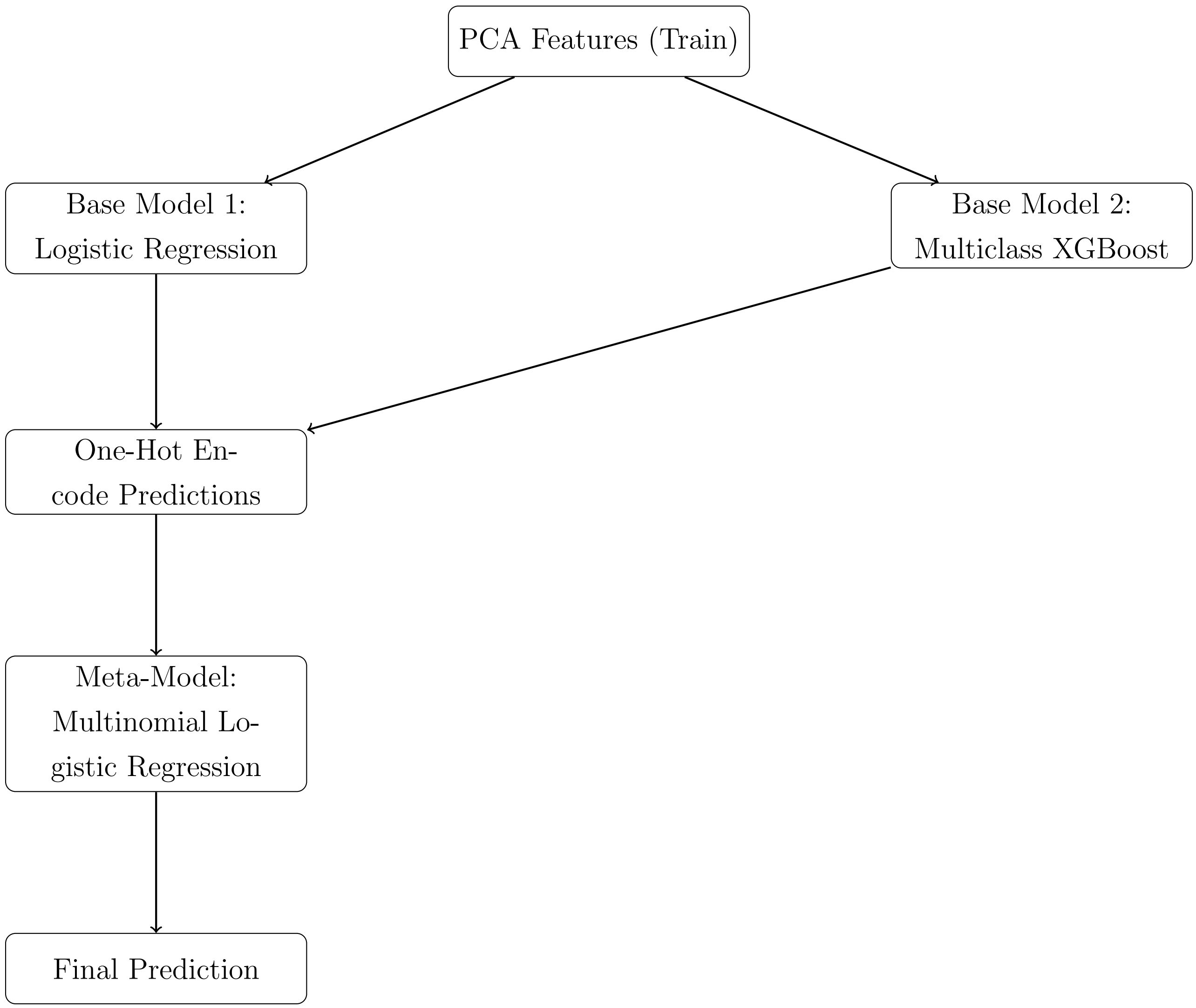


Figure 3: Stacked Ensemble Architecture

Steps Taken to Optimize System Performance and Limit Run Time

I was constantly in a battle between performance (F1 Score) and runtime. Here are the steps I'm taking to optimize both.

PCA Variance (Runtime Fix)

Problem: Using 95% PCA variance made RandomForest take 764 seconds.
Solution: Reduced PCA variance to 92%, lowering feature count and drastically speeding up training.

Model Architecture (F1 Fix)

Problem: Weak boosting models (XGBoostSimple, AdaBoost) got stuck around F1 0.65.
Solution: Replaced them with **MulticlassXGBoost** using full decision trees. This increased F1 from 0.65 to 0.91.

Stacking Model Selection (F1 Fix)

Problem: Stacks including overfit RandomForest failed (F1 0.13).
Solution: Removed "poison" models and kept only LR and XGB that drastically give me the best F1 score.

4. Stacking Bug Fix (F1 Fix)

Problem: Even good models produced F1 0.12 due to feeding integers (0–9) as numeric features.
Solution: One-Hot Encoded model predictions before training the meta-model.

My Thoughts and Learnings from This Project

This project has been a fantastic (and sometimes frustrating!) exercise in debugging and diagnostics. Building the models was the easy part; figuring out *why* they weren't working was the real challenge.

1. The "PCA Ceiling" — My Biggest Insight

I realized my 91% F1 is the performance ceiling of PCA-based data.

- Friends used raw pixels with non-linear XGBoost (can hit 95%)
- I used PCA-transformed linear-optimized features

My LR achieving 0.90 proves my pipeline is excellent; tuning XGBoost to beat LR (0.913) is a major win.

2. "Ensemble" Isn't Magic

A bad model in a stack poisons the meta-model. RandomForest (F1 0.80) misled the stack and caused F1 0.13. Then when i removed , i got the wow moment.

3. Feature Engineering ¿ Model Choice

The OHE bug was the biggest lesson: data representation matters as much as the algorithm.Even after the best algo, my F1 was too low because of that OHE bug

4. Development vs. Production

The `main.py` failures (NameError, missing variables) taught me that:

- Notebooks are for development
- Scripts must be fully self-contained

5. Debugging is the Real Skill

The real project from getting F1 for me changed to diagnosing:

- Why XGBoostSimple was weak
- Why RandomForest overfit
- Why stacking failed (OHE bug)
- Why `main.py` broke (f-string bug)

This project has been a masterclass in diagnostics and perseverance, and I'm very happy with the final, robust pipeline. Because i used PCA ,with 92 variance, thats why my this pipeline can have atmost around 91-92 F1 score but that's also great because even we know that we will be using NeuralNetwork(MultiPerceptron) in real life instead of These classification models.