MNIST Digit Classification System

Final Project Report

Chiru Thejaswi

(a) Summary of Models Used and System Architecture

The goal of this project was to implement a complete MNIST multi-class classification system using only Python, NumPy, and manual implementation of machine learning algorithms, without relying on sklearn or external ML libraries. The system architecture contains the following components:

- **Ridge Regression (One-vs-Rest):** Linear classifier trained using the closed-form ridge solution.
- **Softmax Regression:** Multinomial logistic regression optimized using gradient descent.
- **Nearest Centroid Classifier:** Classifies a sample based on distance to the mean vector of each digit class.
- **Bagged Softmax Models (10 models):** Each model is trained on a bootstrap sample of the training set to reduce variance.
- **RFF (Random Fourier Features) + Softmax:** Uses random projections to approximate an RBF kernel, followed by Softmax.
- **Stacked Ensemble Meta-Learner:** A final ridge regression classifier trained on prediction outputs of all base models.

The final pipeline loads MNIST CSV files, constructs feature matrices, trains all models, evaluates validation metrics, and finally builds an ensemble through stacking.

(b) Hyperparameter Tuning and Experimental Results

Multiple hyperparameters were tuned during experimentation:

- Ridge OVR: regularization strength λ.
- Softmax Regression: learning rate, iterations, and regularization.
- Bagging: number of Softmax models.

- RFF Kernel Model: number of random features and frequency scale.
- Meta-Learner: ridge regularization parameter.

The following table shows the final validation-set results for every model in the system:

Model	Macro-F1	Accuracy
ridge	0.8444	0.8475
softmax	0.8857	0.8872
nearest centroid	0.8432	0.8435
soft_bag_0	0.8867	0.8880
soft_bag_1	0.8831	0.8844
soft_bag_2	0.8893	0.8908
soft_bag_3	0.8904	0.8916
soft_bag_4	0.8885	0.8900
soft_bag_5	0.8929	0.8940
soft_bag_6	0.8854	0.8868
soft_bag_7	0.8915	0.8928
soft_bag_8	0.8897	0.8912
soft_bag_9	0.8890	0.8904
rff	0.7671	0.7675
stacked (final)	**0.9023**	**0.9036**

(c) Runtime Optimization Steps and Performance

Total runtime of the full system was **13.8 seconds**. Several optimizations were implemented to stay well within the required limit:

- **Vectorized Numpy Implementation:** All models use pure NumPy linear algebra.
- **Precomputed Feature Matrices:** RFF features and One-vs-Rest matrices computed once.
- **Efficient Bagging:** Only Softmax is bagged, as it is fast and stable.
- **Closed-Form Solvers:** Ridge OVR and meta-learner use matrix formulas, avoiding iteration.
- **Minimal Feature Engineering:** RFF uses only 100 random features to balance accuracy and speed.

Training and validation F1 scores show that:

- Linear models generalize well but have limited expressiveness.
- Softmax regression is the strongest individual model.
- Bagging consistently improves Softmax performance.
- The stacked ensemble provides the best macro-F1 and accuracy.

(d) Thoughts and Observations

This project reinforced several important concepts:

- **Bias–Variance Tradeoff:** Ridge OVR and Nearest Centroid show high bias, while bagged Softmax significantly reduces variance.
- **Ensemble Benefit:** Over 10 bagged Softmax models, performance steadily improves and stabilizes. This demonstrates how variance reduction helps on real data.
- **RFF Limitations:** Random Fourier Features performed noticeably worse, showing that kernel approximations require larger feature counts to accurately mimic RBF kernels.
- **Stacked Model Superiority:** The meta-learner effectively learns when to trust each base model. It captures complementary decision patterns, achieving the best macro-F1: **0.9023**.
- **Practical Understanding:** Writing algorithms manually (OVR, Softmax, Bagging, RFF) forces deeper understanding of linear algebra, optimization, and the behavior of real-world models.

Overall, the project provided strong practical insight into classical machine-learning methods, ensemble learning, and system-level design, all within strict runtime and library constraints.