DA2401 - Machine Learning Lab

End Semester Assignment
MNIST Digit Classification

Multi-Class Classification System

November 2025

1 Executive Summary

This report presents a multi-class classification system for MNIST digit recognition using ensemble methods. The system achieves an F1 score of 95.02% on the validation dataset using K-Nearest Neighbors (KNN) as the primary classifier, with training time under 5 minutes. Three algorithms were implemented from scratch: Softmax Regression, XGBoost Classifier, and KNN Classifier.

2 Model Summary & System Architecture

2.1 Implemented Algorithms

Three classification algorithms were implemented in plain Python with NumPy:

1. **Softmax Regression with PCA**: A linear classifier using gradient descent optimization with optional Principal Component Analysis for dimensionality reduction.
2. **XGBoost Classifier**: A gradient boosting algorithm using decision trees with second-order gradients (Hessians) for multiclass classification via one-vs-rest approach.
3. **K-Nearest Neighbors (KNN)**: A non-parametric classifier using Euclidean distance metric for majority voting.

2.2 System Architecture

The final system uses KNN with k=5 as the primary classifier due to its superior performance (95.02% F1 score) and zero training time. The architecture is as follows:

- **Input**: 784-dimensional feature vectors (28×28 pixel images flattened)
- **Preprocessing**: Raw pixel values (0-255) used directly, no normalization required
- **Classification**: KNN with k=5 neighbors using Euclidean distance
- **Output**: Predicted digit class (0-9)

2.3 Alternative Approaches Explored

Bagging with Softmax Regression: Implemented bagging ensemble with 10 Softmax models trained on bootstrap samples (80% sample ratio). This improved F1 score from 88% to 90%, but was not included in the final system due to training time constraints.

Hybrid Model (Abandoned): Attempted a two-stage system using XGBoost as primary classifier with Softmax for correcting commonly misclassified labels. This approach failed due to severe class imbalance in the one-vs-all Softmax models (only 10% positive samples), resulting in worse F1 scores.

3 Hyperparameter Tuning & Results

3.1 Softmax Regression

3.1.1 Hyperparameters Explored

Parameter	Range Tested	Final Value
Learning Rate	0.01 - 0.5	0.1
Epochs	100 - 2000	1500
PCA Components	20 - 784	40

Table 1: Softmax Regression Hyperparameters

3.1.2 Observations

- **Learning Rate**: Low values (¡ 0.05) with insufficient epochs led to poor convergence (F1 score 76-80%). High values (¿ 0.3) caused overshooting and convergence to suboptimal local minima.
- **Epochs**: Models required at least 1000 epochs to converge properly. Final loss at epoch 1999 was 1.883104.
- **PCA Impact**: No significant difference in performance or training time between PCA-reduced (40 components) and full-dimensional (784 features) data. Both achieved similar F1 scores of 87-88%.

3.1.3 Performance

- **Final F1 Score**: 87.37% (validation set)
- **Training Loss**: 1.883104 (final epoch)
- **Training Time**: Approximately 45-60 seconds

3.1.4 Misclassification Analysis

Most common misclassifications for Softmax:

True Class	Predicted As	Count
0	-	19
1	-	15
2	-	39
3	-	33
4	-	14
5	-	35
6	-	27
7	-	17
8	-	42
9	-	77

Table 2: Misclassification counts for Softmax Regression

Class 9 had the highest misclassification rate (77 errors), indicating difficulty in distinguishing this digit.

3.2 Bagging Ensemble with Softmax

3.2.1 Hyperparameters

Parameter	Value
Number of Estimators	10
Sample Ratio	0.8
Learning Rate	0.1
Epochs per Model	1000
PCA Components	40
Random State	42

Table 3: Bagging Ensemble Parameters

3.2.2 Performance

- **Soft Voting F1 Score**: 89.76%
- **Hard Voting F1 Score**: 89.72%
- **Improvement**: +2.4% over single Softmax model
- **Training Time**: Approximately 5-8 minutes (too slow for final system)

Final losses for individual models ranged from 0.737 to 2.828, showing variance in bootstrap sample quality.

3.3 XGBoost Classifier

3.3.1 Hyperparameters

Parameter	Value
Number of Estimators	50
Learning Rate	0.3
Max Depth	2
Min Samples Split	4
Gamma	0.0
Regularization Lambda	0.1
Column Sample Ratio	0.8
PCA Components	40
Random State	11

Table 4: XGBoost Hyperparameters

3.3.2 Training Progress

Trees	Loss	Accuracy
10	0.594045	0.8432
20	0.379594	0.9019
30	0.281420	0.9300
40	0.216990	0.9478
50	0.172516	0.9635

Table 5: XGBoost Training Progress

3.3.3 Performance

- **Final F1 Score**: 90.51% (validation set)
- **Training Accuracy**: 96.35%
- **Training Loss**: 0.172516
- **Training Time**: Approximately 4-5 minutes

3.3.4 Misclassification Analysis

True Class	Most Confused With	Count
0	5	3
1	2	4
2	4	6
3	5	9
4	9	11
5	3	11
6	5	9
7	9	8
8	5	10
9	4	13

Table 6: XGBoost Misclassification Patterns

Common confusion pairs: (4, 9) and (3, 5) due to structural similarity.

3.3.5 Design Choices

- **Max Depth = 2**: Balanced between model complexity and training time. Deeper trees significantly increased training time beyond 5-minute limit.
- **50 Estimators**: Trade-off between performance and training time. More trees would improve accuracy but exceed time constraints.
- **PCA**: Slight reduction in training time with minimal accuracy impact.

3.4 K-Nearest Neighbors (KNN)

3.4.1 Hyperparameters

Parameter	Value
k (neighbors)	5
Distance Metric	Euclidean

Table 7: KNN Hyperparameters

3.4.2 Performance

- **Final F1 Score**: 95.02% (validation set)
- **Training Time**: 0 seconds (lazy learner)
- **Prediction Time**: 30-40 seconds for 2499 validation samples

3.4.3 Hyperparameter Selection

K-value experimentation showed that k=5 provided the best validation performance. Higher k values led to increased errors due to including distant neighbors in voting.

3.4.4 Misclassification Analysis

True Class	Most Confused With	Count
0	6	2
1	-	0
2	1	5
3	7	5
4	9	7
5	6	3
6	0	4
7	1	4
8	3	8
9	4	7

Table 8: KNN Misclassification Patterns

Class 1 had perfect classification on validation set. Common confusion pairs remained (4, 9) and (0, 6).

4 Performance Optimization & Runtime

4.1 Strategies to Limit Training Time

4.1.1 Algorithm Selection

KNN as Primary Classifier: Zero training time since KNN is a lazy learner. All computation occurs during prediction phase, which is acceptable as prediction time is not constrained.

4.1.2 XGBoost Optimizations

- **Reduced Tree Depth**: Limited max_depth to 2 instead of deeper trees (3-5), cutting training time by approximately 40-50%.
- **Limited Estimators**: Used 50 trees instead of 100+ trees typical in production systems.
- **Column Subsampling**: 80% feature sampling per split reduced computation in best-split search.
- **PCA Dimensionality Reduction**: Reduced features from 784 to 40, providing slight speed improvement.

4.1.3 Softmax Optimizations

- **Vectorized Operations**: Used NumPy broadcasting for gradient computation across all samples simultaneously.
- **PCA Trade-off**: PCA showed minimal benefit for Softmax, so both compressed (40 components) and full-dimensional (784 features) versions had similar performance.

4.1.4 Abandoned Approaches

- **Bagging**: While improving F1 score by 2.4%, training 10 models exceeded time budget (8-10 minutes total).

- **Hybrid Models**: Two-stage systems with XGBoost + Softmax correction added computational overhead without accuracy gains.

4.2 Evaluation Results

4.2.1 Training Dataset Performance

Model	F1 Score	Training Time	Status
Softmax Regression	87.37%	45-60s	Alternative
Softmax + Bagging	89.76%	8-10m	Too Slow
XGBoost	90.51%	4-5m	Alternative
KNN (k=5)	-	0s	**Selected**

Table 9: Model Comparison - Training Phase

4.2.2 Validation Dataset Performance

Model	F1 Score	Prediction Time
Softmax Regression	87.37%	¡1s
Softmax + Bagging	89.76%	5s
XGBoost	90.51%	2s
KNN (k=5)	**95.02%**	30-40s

Table 10: Model Comparison - Validation Results

4.3 Final System Timing

- **Total Training Time**: 0 seconds (KNN)
- **Prediction Time (2499 samples)**: 35 seconds
- **Total Runtime**: ¡1 minute (well under 5-minute constraint)

5 Detailed Observations & Insights

5.1 Common Misclassification Patterns

All three models exhibited similar confusion patterns:

- **4 vs 9**: Structural similarity due to curved upper portion
- **0 vs 6**: Both contain circular shapes
- **3 vs 5**: Similar curved regions

This suggests that ensemble methods combining different algorithms may not significantly improve performance, as all models struggle with the same inherent feature ambiguities.

5.2 Why Ensemble Methods Failed

Hypothesis: Ensemble methods (bagging, boosting, stacking) work best when base learners make different types of errors. In MNIST classification:

- All models use similar feature representations (raw pixel values)
- Confusion occurs on structurally similar digits
- Combining predictions doesn't resolve fundamental ambiguity

Evidence from Hybrid Model Failure: The two-stage XGBoost + Softmax system performed worse because:

1. Softmax models trained on imbalanced data (10% positive class)
2. One-vs-rest approach created unreliable probability estimates
3. Correction stage introduced new errors (e.g., misclassifying 9 as 7)

5.3 PCA Analysis

Surprising Finding: PCA showed minimal impact on both performance and training time:

- **Softmax**: No significant F1 score difference between 40 and 784 components
- **XGBoost**: Slight speed improvement but negligible accuracy change
- **KNN**: Not tested due to zero training time advantage

Interpretation: MNIST images are already low-resolution (28×28), and digit structure requires most pixel information. Aggressive dimensionality reduction (to 40 components) retains sufficient variance for classification.

5.4 Training Time vs Accuracy Trade-off

Model	Time Investment	F1 Score
Softmax (simple)	1 minute	87%
Softmax + Bagging	10 minutes	90%
XGBoost (optimized)	5 minutes	90.5%
KNN	0 minutes	95%

Table 11: Time-Accuracy Trade-off

Key Insight: KNN achieves highest accuracy with zero training time, making it optimal for this constrained scenario. The 4.5% improvement over XGBoost comes "for free" in terms of training budget.

5.5 Limitations of Implementation

- **KNN Prediction Speed**: While training is instant, prediction requires computing distances to all 10,002 training samples for each test point. This is acceptable for small test sets but would not scale to production.
- **No Feature Engineering**: All models used raw pixel values. Domain-specific features (e.g., stroke direction, loop detection) were not explored due to time constraints.
- **Hyperparameter Search**: Manual tuning rather than systematic grid search. Optimal parameters may exist but were not found.

5.6 Lessons Learned

1. **Algorithm Selection Matters More Than Complexity**: Simple KNN outperformed sophisticated ensemble methods.
2. **Training Time Constraints Favor Lazy Learners**: When training time is limited but prediction time is flexible, KNN-style algorithms excel.
3. **Ensemble Methods Require Diversity**: Combining models that make similar errors provides minimal benefit.
4. **Validation Set Insights**: Misclassification analysis revealed that structural digit similarity is the primary challenge, not model sophistication.
5. **PCA Not Always Beneficial**: For already low-dimensional data (784 features for 10,002 samples), dimensionality reduction may be unnecessary.

6 Conclusion

The final classification system uses K-Nearest Neighbors with k=5, achieving 95.02% F1 score on the validation dataset with zero training time. This exceeds the performance of more complex models (XGBoost: 90.51%, Softmax: 87.37%) while meeting the 5-minute training time constraint.

The key insight from this exercise is that algorithm appropriateness for the problem context (small dataset, structural pattern recognition) matters more than algorithmic complexity. KNN's non-parametric nature and implicit learning of complex decision boundaries proved superior to linear models (Softmax) and explicit feature engineering (XGBoost trees).

Future improvements could explore:

- Distance metric optimization (e.g., learned metrics)
- Approximate nearest neighbor algorithms for faster prediction
- Feature engineering to resolve structural ambiguities (4 vs 9, 0 vs 6)