DevSoc Core Assignments 2025-26

Kamal Kumar Manchenella

June 2025

Contents

1 Ensemble Learning

1.1 Overview of Ensemble Learning

Ensemble learning is a machine learning paradigm where multiple base models are strategically generated and combined to solve a particular computational intelligence problem. The main idea is that a group of weak learners can come together to form a strong learner.

There are three main types of ensemble methods:

- **Bagging (Bootstrap Aggregating)**: Trains multiple versions of a model on different random subsets of the training data and averages their predictions. Example: Random Forest.
- **Boosting**: Trains models sequentially, where each model tries to correct the errors of its predecessor. Later models focus more on the hard-to-classify examples. Examples: Gradient Boosting, XGBoost.
- **Voting**: Combines predictions from multiple models. In *soft voting*, the average of predicted class probabilities is used; in *hard voting*, the most frequent class label among the base models is chosen.

These techniques increase robustness and accuracy by reducing variance, bias, or improving prediction stability.

1.2 Objective

The goal of this project was to predict an individual's obesity category based on health, lifestyle, and demographic indicators using machine learning. The problem was formulated as a multi-class classification task and addressed using various ensemble learning techniques to improve performance over traditional models.

1.3 Dataset and Features

The dataset used contains 2111 records and 17 columns, including features such as `Age`, `Height`, `Weight`, `Physical Activity (FAF)`, `Daily Water Intake (CH2O)`, and categorical variables like `Gender`, `Smoking`, and `Family History`. The target variable was `NObeyesdad`, representing one of seven obesity categories.

1.4 Preprocessing

All categorical features were label-encoded or one-hot encoded as appropriate. Numerical features were scaled using standardization. The data was then split into 80% training and 20% testing subsets. The target class distribution was balanced and stratified.

1.5 Models Used

We implemented and evaluated the following models:

- Logistic Regression (Baseline)
- Random Forest (Bagging)
- Gradient Boosting
- XGBoost
- Voting Ensemble (Soft)

Confusion Matrix Terminology

Term	**Meaning**
TP (True Positive)	Correctly predicted positive class
TN (True Negative)	Correctly predicted negative class
FP (False Positive)	Incorrectly predicted as positive
FN (False Negative)	Incorrectly predicted as negative

Table 1: Basic terms used in confusion matrices

1.6 Evaluation Metrics

Performance was evaluated using:

- Accuracy
- Macro-Averaged F1 Score
- Confusion Matrices (for class-wise comparison)

Metric Definitions

The following standard classification metrics were used:

- **Accuracy**:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**:

$$Precision = \frac{TP}{TP + FP}$$

- **Recall**:

$$Recall = \frac{TP}{TP + FN}$$

- **F1 Score**:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

- **Macro F1 Score**:

$$Macro - F1 = \frac{1}{C} \sum_{i=1}^{C} F1_i$$

where C is the number of classes.

1.7 Results and Discussion

Figure 1 shows the performance comparison across all models in terms of Accuracy and Macro F1 Score.

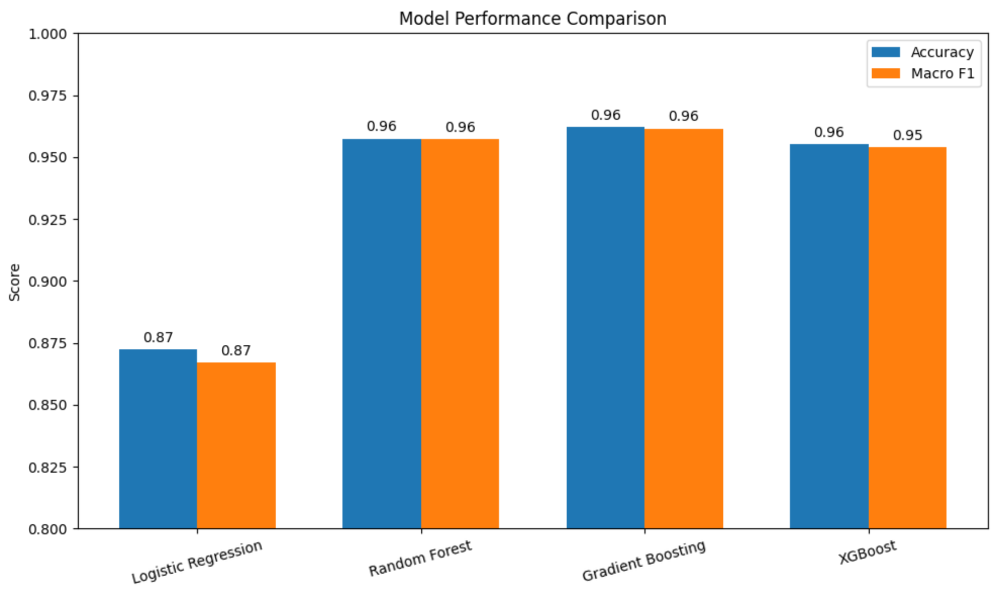


Figure 1: Accuracy and F1 Score Comparison across Models

Baseline (Logistic Regression): Achieved an accuracy of 87%. However, it struggled with mid-range classes like `Overweight_Level_I`, as seen in Figure **??**.
We evaluated four individual models before ensembling: Logistic Regression (baseline), Random Forest, Gradient Boosting, and XGBoost. All ensemble-based models outperformed the baseline, achieving high classification performance with 96% accuracy on average. Figure 6 presents their confusion matrices for comparison.

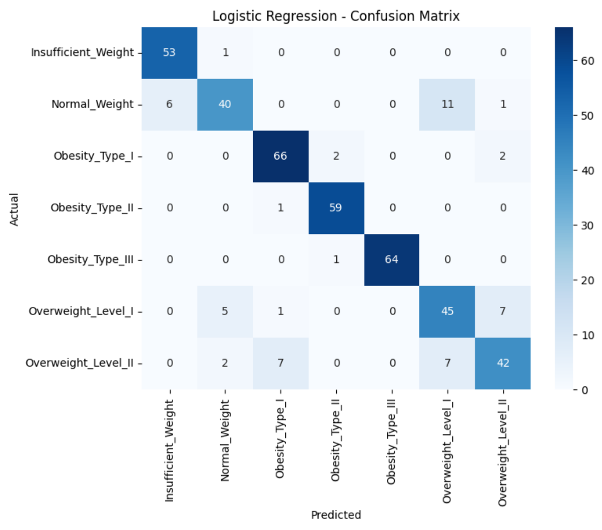


Figure 2: *
Logistic Regression

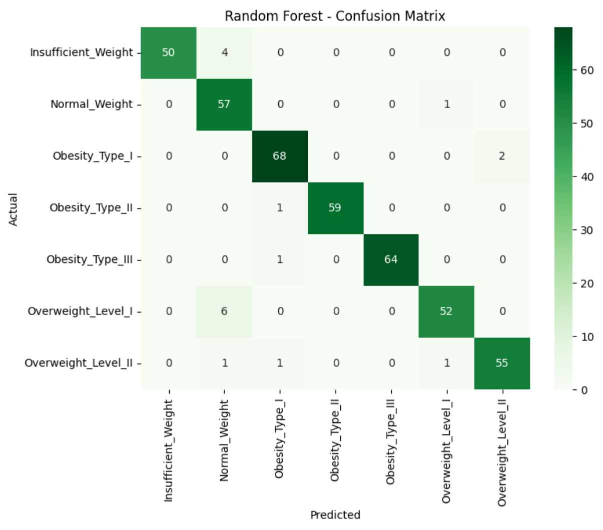


Figure 3: *
Random Forest

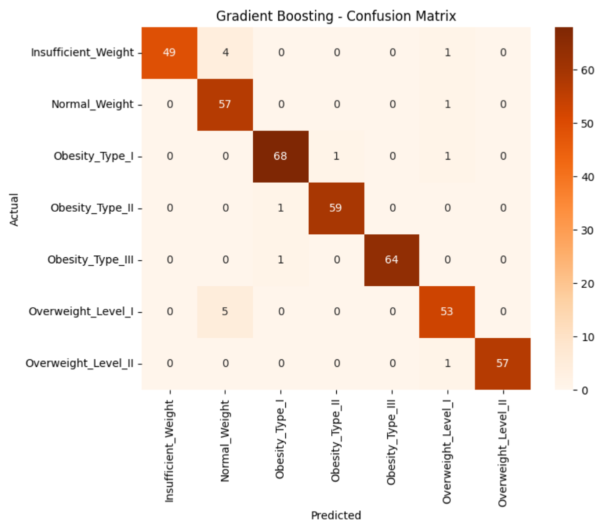


Figure 4: *
Gradient Boosting

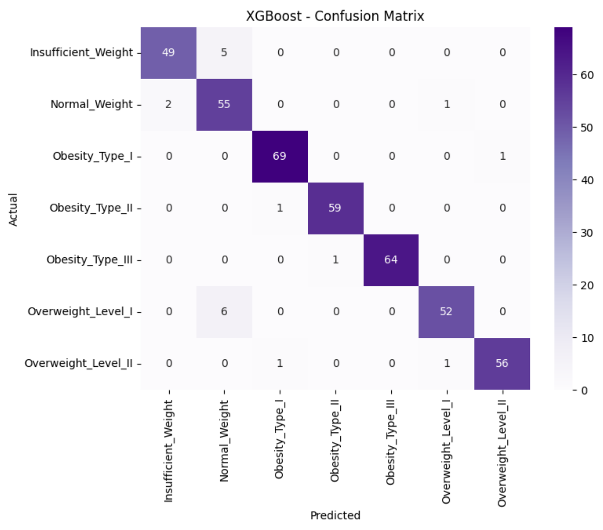
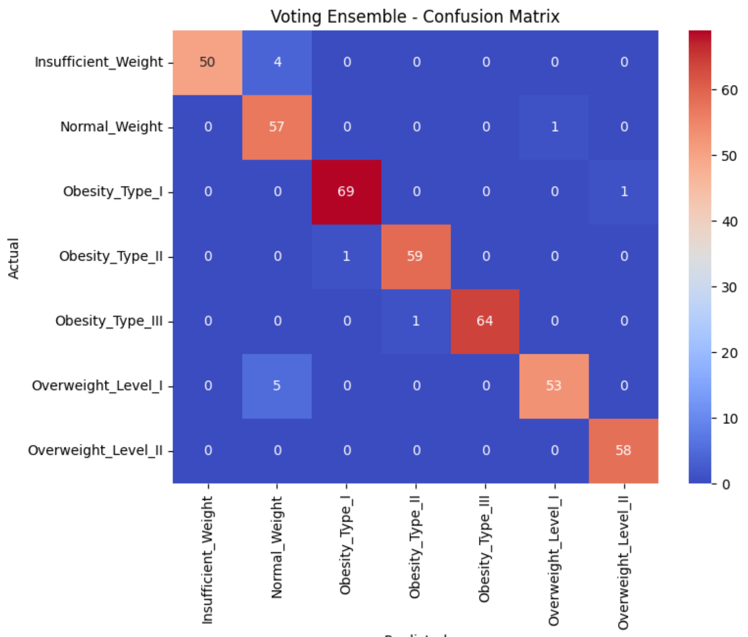


Figure 5: *
XGBoost

Figure 6: Confusion Matrices for Singular Models: Logistic Regression, Random Forest, Gradient Boosting, and XGBoost



Actual \ Predicted	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_II	Obesity_Type_III	Overweight_Level_I	Overweight_Level_II
Insufficient_Weight	50	4	0	0	0	0	0
Normal_Weight	0	57	0	0	0	1	0
Obesity_Type_I	0	0	69	0	0	0	1
Obesity_Type_II	0	0	1	59	0	0	0
Obesity_Type_III	0	0	0	1	64	0	0
Overweight_Level_I	0	5	0	0	0	53	0
Overweight_Level_II	0	0	0	0	0	0	58

Figure 7: Voting Ensemble - Confusion Matrix

1.8 Conclusion

Ensemble models, particularly the Voting Classifier, significantly outperformed the baseline. This highlights the robustness of combining multiple algorithms in structured healthcare classification tasks. Future work could include hyperparameter optimization, feature importance ranking, and deployment using model serialization.

2 MNIST Neural Network

2.1 Overview

This project demonstrates the implementation of a simple feedforward neural network (also known as a Multi-Layer Perceptron or MLP) from scratch in Python using only NumPy. The objective is to classify handwritten digits (0–9) from the MNIST dataset. The network is trained using Stochastic Gradient Descent (SGD) with cross-entropy loss, and all aspects—forward pass, backpropagation, and weight updates—are coded manually.

2.2 Dataset

The MNIST dataset consists of 70,000 grayscale images of handwritten digits, each of size 28×28. The training set contains 60,000 images and the test set contains 10,000 images. Each image is flattened into a 784-dimensional vector and the labels are one-hot encoded for classification.

2.3 Network Architecture

The neural network has the following structure:

Layer	**Configuration**
Input Layer	784-dimensional vector (flattened 28×28 image)
Hidden Layer	$784 \rightarrow 128$, ReLU activation
Output Layer	$128 \rightarrow 10$, Softmax activation

Table 2: Fully Connected Neural Network Architecture for MNIST Digit Classification

- Input layer: 784 neurons (one for each pixel)
- Hidden layer: 128 neurons with ReLU activation
- Output layer: 10 neurons with Softmax activation

2.4 Mathematical Formulation

$\mathrm{Z}^{[1]} = XW^{[1]} + b^{[1]}$
$A^{[1]} = ReLU(Z^{[1]})$
$Z^{[2]} = A^{[1]}W^{[2]} + b^{[2]}$
$\hat{Y} = Softmax(Z^{[2]})$

The loss function used is categorical cross-entropy:

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{C} Y_{ij}\log(\hat{Y}_{ij})$$

where m is the number of samples and C is the number of classes.

2.5 Training Details

- Optimizer: Stochastic Gradient Descent (SGD)
- Batch Size: 64
- Learning Rate: 0.1
- Epochs: 10

2.6 Evaluation Metrics

Model performance was evaluated using:

- Accuracy
- Confusion Matrix
- Visual Inspection of Predictions

2.7 Results

Epoch	**Train Loss**	**Test Loss**	**Test Accuracy (%)**
1	0.0437	0.0769	97.79
2	0.0396	0.0721	97.79
3	0.0388	0.0746	97.81
4	0.0359	0.0729	97.76
5	0.0307	0.0712	97.83
6	0.0277	0.0679	97.86
7	0.0263	0.0682	97.87
8	0.0241	0.0701	97.91
9	0.0220	0.0656	97.89
10	0.0220	0.0720	97.78

Table 3: Improved epoch-wise training and testing performance of the neural network on MNIST. The model continues to generalize well and maintain high accuracy.

Final test accuracy: **97.78%**

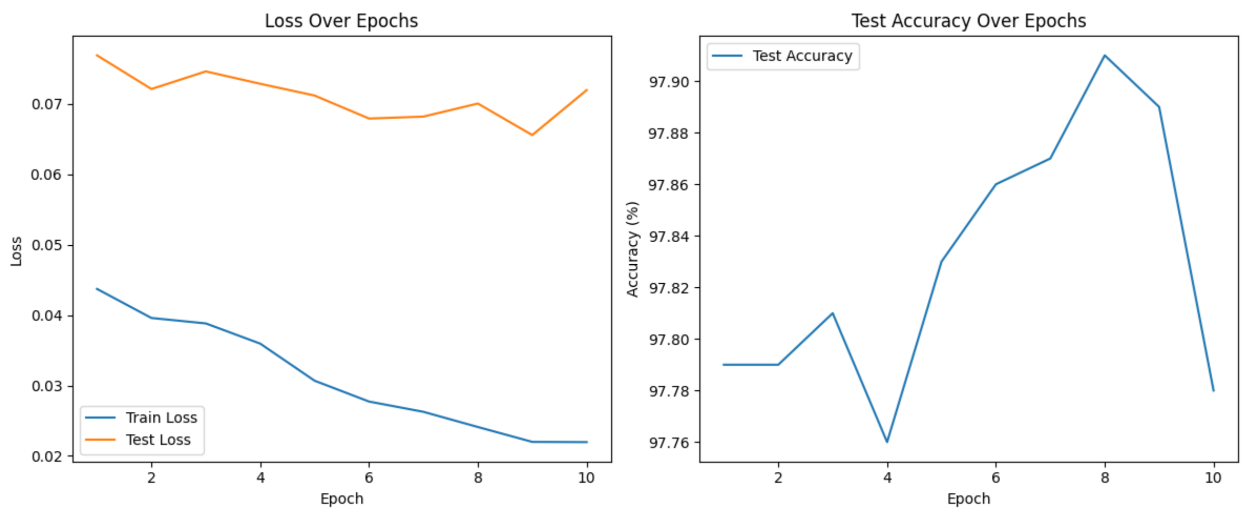


Figure 8: Visualization of training loss, test loss, and test accuracy over 10 epochs. The network demonstrates steady convergence and generalization.

2.8 Visualizations

- **Confusion Matrix:** Demonstrated strong class-wise accuracy with minimal confusion
- **Weight Visualization:** First-layer weights were reshaped into 28×28 grids to interpret learned features
- **Sample Predictions:** Model predictions over randomly selected test samples were accurate and matched ground truth

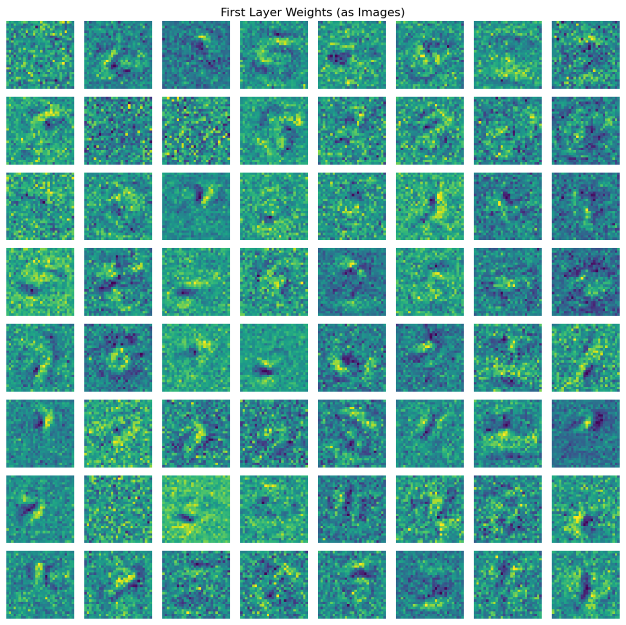


Figure 9: Visualization of the first layer weights reshaped to 28×28 patches. These filters show how the model learns digit structures and edges.

2.9 Conclusion

This project showcases the core mathematical and computational principles of training a neural network. Achieving near state-of-the-art accuracy using only NumPy validates the correctness of the implementation. This also builds intuition for deeper models, activation functions, and optimization. Future enhancements could include regularization, learning rate scheduling, and implementation of momentum-based optimizers.