Ensemble Learning for Early Flood Warning System in Wastewater Networks

1st Sampreet Ajjanagouda Patil
Computer Science Dept.
Texas A&M University-Corpus Christi
Corpus Christi, USA
spatil1@islander.tamucc.edu

2nd Gowtham Kilaru
Computer Science Dept.
Texas A&M University-Corpus Christi
Corpus Christi, USA
gkilaru@islander.tamucc.edu

3rd Khaja Faizan Mohammad
Computer Science Dept.
Texas A&M University-Corpus Christi
Corpus Christi, USA
kmohammad@islander.tamucc.edu

4th Mohammed Taha Ayub Khan
Computer Science Dept.
Texas A&M University-Corpus Christi
Corpus Christi, USA
mkhan30@islander.tamucc.edu

***Abstract*—This project develops an ensemble machine learning framework to detect flood events in wastewater networks by predicting water depth thresholds. Utilizing datasets of flow depth, flow rate, and node features, we implement and evaluate five models: Neural Networks, Random Forests, Gradient Boosting Trees, Linear Regression, and Support Vector Machines. An ensemble approach integrates these models to enhance predictive performance. Results show high accuracy and F1 scores, with the Neural Network achieving 0.9972 accuracy and 0.9845 F1, and the ensemble yielding a robust 0.9634 F1 score. This system provides a scalable, real-time solution for urban flood monitoring, addressing the limitations of traditional hydraulic models.[1]**

CONTENTS

I. INTRODUCTION

Urban wastewater networks are critical infrastructure components that face significant challenges during extreme weather events, particularly heavy rainfall.[1] Such conditions can overwhelm system capacity, leading to flooding that threatens public health, damages infrastructure, and disrupts daily life in cities. For instance, overflow events can contaminate water supplies, erode roadways, and inundate residential areas, creating hazardous conditions that require immediate response. Traditional hydraulic models provide detailed simulations of water flow and depth.[2] However, these models are computationally intensive, requiring extensive setup and processing time, which renders them impractical for real-time flood detection and early warning systems in dynamic urban environments.[3]

This project addresses these challenges by proposing a machine learning-based flood detection system tailored for wastewater networks. Our primary objective is to predict whether water depth at network nodes exceeds a critical threshold (set at 0.5 units in this study), thereby triggering an early warning signal for flood risk. Unlike hydraulic models, machine learning approaches can process large volumes of historical and real-time data quickly, offering a scalable alternative for urban flood management.[2]We employ an ensemble of diverse models—Neural Networks (NN), Random Forests (RF), Gradient Boosting Trees (GBT), Linear Regression (LR), and Support Vector Machines (SVM)—to achieve a balance of accuracy, interpretability, and computational efficiency. Each model contributes unique strengths, from capturing non-linear patterns to providing robust generalization, enhancing overall prediction reliability.

The motivation for this work is rooted in the increasing demand for smart infrastructure solutions as cities grow and climate change intensifies rainfall patterns.[3] Single-model approaches, while effective in controlled settings, often suffer from limitations such as overfitting, sensitivity to noise, or in-ability to handle diverse data types (e.g., time-series and spatial features). By integrating multiple machine learning techniques into an ensemble, we aim to mitigate these issues and provide a more resilient system. This report comprehensively details our methodology, including data preprocessing, model development, and training strategies, followed by experimental results and insights into Says into practical deployment. Our findings demonstrate the potential of ensemble learning to revolutionize flood monitoring, paving the way for proactive urban resilience strategies.[4]

II. Technologies Used

The project leverages a robust and carefully selected stack of technologies to support data processing, machine learning implementation, and result analysis:

- **Python**: As the primary programming language, Python is chosen for its versatility, extensive library ecosystem, and widespread adoption in data science and machine learning communities. Its readability and flexibility facilitate rapid development and debugging, critical for iterative experimentation.
- **Pandas & NumPy**: These libraries form the backbone of data manipulation and numerical computation. Pandas provides powerful data structures (e.g., DataFrames) for handling structured datasets, while NumPy supports efficient array operations, enabling fast preprocessing and feature engineering on large-scale data.[1]
- **PyTorch**: A leading deep learning framework, PyTorch is utilized for constructing and training the Neural Network model. Its dynamic computation graph allows for flexible architecture design, and its optimization tools (e.g., Adam optimizer) support efficient gradient-based learning. PyTorch's GPU acceleration capability further enhances training speed, making it ideal for our deep learning component.[4]
- **Jupyter Notebook**: This interactive development environment integrates code execution, visualization, and documentation in a single interface. It enables step-by-step analysis of data preprocessing, model training, and evaluation, with inline outputs (e.g., loss curves, performance metrics) that enhance transparency and reproducibility.

Together, these tools create a cohesive workflow, from raw data ingestion to model deployment, aligning with the project's goals of accuracy, efficiency, and scalability.[1] Their open-source nature also ensures accessibility for future extensions or collaborative efforts.[5]

III. Data Description

The dataset is composed of three distinct CSV files, each contributing critical information for flood detection:

- **Flow Depth (Flow_depth_v3.csv)**: This file contains hourly time-series data capturing water depth measurements across multiple nodes in the wastewater network. Columns include timestamps (originally labeled 'IDs:' or 'Date/Time') and depth values for various nodes, providing a temporal view of water level fluctuations.
- **Flow Rate (Flow_rate_v3.csv)**: Complementary to the flow depth data, this file records flow rates at the same nodes, aligned by timestamp. It captures the volume of water moving through the system, a key indicator of potential overflow conditions.
- **Node Features (WW01_node.csv)**: This dataset provides static attributes of network nodes, including Node ID, X/Y coordinates (spatial positioning), and invert elevation (in feet, representing the lowest point of the pipe). These features offer contextual information about the physical layout and capacity of the network.

After preprocessing, the merged dataset comprises 17,707 samples with 49 features, including derived temporal attributes (hour, day, month) extracted from timestamps. Normalization to a [0,1] range ensures uniformity across features, mitigating biases from differing scales (e.g., depth vs. elevation). The resulting dataset, while rich in temporal and measurement data, lacks full integration with node features due to a preprocessing error, which we address in the methodology section.

IV. Methodology

A. Data Preprocessing

Effective preprocessing is foundational to machine learning success, ensuring data quality and compatibility with model requirements. Our process involves several detailed steps:

1) **Column Renaming**: We standardize column names for consistency, renaming 'IDs:' or 'Date/Time' to 'Time' across flow depth and rate datasets. This simplifies merging and ensures uniform referencing throughout the pipeline.
2) **Type Conversion**: Timestamps are converted from strings to datetime objects using Pandas' `to_datetime`, with errors coerced to NaN for robustness. We extract temporal features—hour, day, and month—to capture cyclic patterns (e.g., diurnal or seasonal effects) that may influence flooding.
3) **Numeric Handling**: Measurement columns (depth and rate) are converted to numeric types with `pd.to_numeric`, handling non-numeric entries by setting them to NaN. Missing values are then imputed with the median of each column, a robust strategy that preserves data distribution under potential outliers.
4) **Normalization**: Features are scaled to $[0, 1]$ using min-max normalization: $X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}} + 10^{-8}}$, where the small constant prevents division by zero. This step ensures all features contribute equally to model training, avoiding dominance by larger-scale variables like elevation.
5) **Merging**: We merge flow depth and rate DataFrames on 'Time' with suffixes ('_depth', '_rate') to distinguish overlapping columns. An attempt to merge with node features on 'NodeID' fails due to its absence in the merged DataFrame, a limitation traced to the initial data structure and flagged for correction.

The output, `data_cleaned`, is a normalized feature matrix of shape $(17{,}707, 49)$, ready for model input but missing spatial context from node features.

B. Model Development

We design five distinct models, each leveraging different learning paradigms to address the binary classification task (depth ¿ 0.5 = 1, else 0):

1) **Neural Network (NN)**: A multilayer perceptron implemented in PyTorch with an input layer (49 features), a hidden layer (64 units, ReLU activation), and an output layer (2 units for binary classification). It uses CrossEntropyLoss to optimize class probabilities, capturing nonlinear temporal and feature interactions.
2) **Random Forest (RF)**: A custom PyTorch-based ensemble of decision trees, where each tree independently predicts class probabilities. The final output aggregates votes, offering robustness to noise and feature redundancy.
3) **Gradient Boosting Tree (GBT)**: A boosting model that iteratively improves weak learners (decision trees), emphasizing misclassified samples. It balances bias and variance, excelling in structured data scenarios like ours.
4) **Linear Regression (LR)**: Adapted as a classifier by outputting logits for two classes, this baseline model assumes linear relationships between features and flood risk, providing simplicity and interpretability.
5) **Support Vector Machine (SVM)**: A margin-based classifier implemented in PyTorch, approximating a kernel method to separate classes with a maximal margin. It handles complex boundaries despite the high-dimensional feature space.

These models collectively span deep learning, tree-based ensembles, and classical methods, ensuring diverse perspectives on the data.

C. Training and Optimization

Training splits the dataset into 80% training (14,165 samples) and 20% testing (3,542 samples) sets, shuffled with a fixed seed (42) for reproducibility:

- **NN**: Trained for 200 epochs with the Adam optimizer (learning rate 0.01 default), with loss monitored every 10 epochs. Grid search explores hidden dimensions (32, 64, 128), learning rates (0.001, 0.01, 0.1), and epochs (50, 100, 200) to maximize F1 score.[2]
- **RF & GBT**: Optimized over number of estimators (50, 100, 200) and learning rates (0.001, 0.01, 0.1), using 200 epochs as a baseline. RF aggregates tree outputs, while GBT boosts sequentially, both leveraging PyTorch's optimization framework.[3]
- **LR & SVM**: Tuned with learning rates (0.001, 0.01, 0.1) over 200 epochs, adapting traditional algorithms to PyTorch's gradient descent paradigm for consistency.
- **Ensemble**: Combines model outputs with weights (0.1 to 0.5), tested exhaustively via grid search across all combinations to identify the optimal weighting scheme.[1]

Evaluation metrics include accuracy (for NN, reflecting correct predictions) and F1 score (for all models, balancing precision and recall), crucial for flood detection where false negatives (missed floods) are costly.

V. Experiments

A. Setup

Experiments are conducted on a dataset of 17,707 samples and 49 features, processed in a Jupyter Notebook environment (e.g., Google Colab). Data is converted to PyTorch tensors (X: float32, y: float32), with training and testing splits executed on CPU or GPU depending on availability. A fixed random seed ensures consistent results across runs.

B. Results

TABLE I
MODEL PERFORMANCE METRICS

Model	Accuracy	F1 Score
Neural Network	0.9972	0.9845
Random Forest	-	0.9329
Gradient Boosting	-	0.9861
Linear Regression	-	0.9205
SVM	-	0.9398
Ensemble	-	0.9634

The Neural Network excels with 0.9972 accuracy and 0.9845 F1 after 200 epochs, reflecting its ability to fit complex patterns. Gradient Boosting achieves the highest F1 (0.9861), suggesting superior recall and precision balance. RF (0.9329), SVM (0.9398), and LR (0.9205) perform respectably, while the ensemble (0.9634) offers a robust compromise. Accuracy is only reported for NN due to explicit notebook output; F1 scores dominate as the primary metric.

C. Best Parameters

- **NN**: Hidden dim = 32, lr = 0.1, epochs = 50 (F1 = 0.9877), outperforming the default 200-epoch run, indicating efficiency at higher learning rates.
- **RF**: n_estimators = 200, lr = 0.1 (F1 = 0.9329), leveraging maximum trees for stability.
- **GBT**: n_estimators = 200, lr = 0.1 (F1 = 0.9861), mirroring RF's optimal settings with boosting benefits.
- **LR**: lr = 0.1 (F1 = 0.9205), simplest yet least effective due to linearity assumptions.
- **SVM**: lr = 0.1 (F1 = 0.9398), benefiting from aggressive optimization.
- **Ensemble**: Weights = (0.5, 0.1, 0.1, 0.1, 0.1) (F1 = 0.9634), heavily favoring NN's contribution.

Grid search consistently favors higher learning rates (0.1), balancing speed and convergence across models.[4]

D. Training Dynamics

The NN's training loop reveals rapid loss reduction, from 0.2933 at epoch 10 to 0.0096 at epoch 200, with significant gains by epoch 50 (0.0440). This suggests effective learning of temporal and feature patterns, corroborated by its high test accuracy. RF and GBT, while lacking epoch-wise logs in the notebook, likely converge similarly due to their tree-based nature, with GBT's boosting providing an edge. LR and SVM, as simpler models, stabilize quickly but lack the capacity for deep pattern recognition.[5]

VI. Discussion

The results underscore the strengths and trade-offs of each approach. The Neural Network's near-perfect accuracy (0.9972) and strong F1 (0.9845) highlight its capacity to model non-linear relationships in time-series data, likely driven by ReLU activation and sufficient hidden units.[4] Gradient Boosting's top F1 (0.9861) reflects its iterative refinement, excelling in handling class imbalance and feature interactions, a common strength of boosting methods. Random Forest (0.9329) offers robustness to noise, while SVM (0.9398) captures complex boundaries, though both trail NN and GBT. Linear Regression (0.9205), as a baseline, struggles with non-linearities, underscoring the dataset's complexity.[2]

The ensemble (F1 = 0.9634), with a 0.5 weight on NN, leverages this model's strength while incorporating others for stability.[3] Its performance, though not the highest, suggests resilience across varied conditions, a key advantage in real-world deployment. However, limitations are evident. The failed 'NodeID' merge excludes spatial context (e.g., elevation, coordinates), potentially reducing model awareness of network topology. The fixed 0.5 threshold oversimplifies flood criteria, ignoring node-specific capacities or regional norms. Computational costs also vary—NN's 200 epochs demand more resources than LR's simplicity—posing trade-offs for real-time use.[4]

Practically, this system could integrate with IoT sensors, feeding live depth and rate data into a cloud-based predictor. High F1 scores ensure reliable alerts, critical for municipal response teams to deploy pumps or close flood-prone areas. Noise sensitivity, untested here, remains a concern for field data, suggesting a need for robustness checks. Scalability to larger networks or different climates also requires validation, potentially with transfer learning or synthetic data augmentation.[3]

VII. Conclusion

This project demonstrates a powerful ensemble machine learning approach for flood detection in wastewater networks, achieving exceptional accuracy (NN: 0.9972) and F1 scores (GBT: 0.9861, Ensemble: 0.9634). By combining deep learning (NN), tree-based ensembles (RF, GBT), and classical methods (LR, SVM), we ensure robust, scalable predictions that outperform traditional hydraulic models in speed and practicality. The system's ability to process time-series data and predict depth exceedances positions it as a viable tool for real-time urban flood management.

Future enhancements include resolving the 'NodeID' merge issue to incorporate spatial features, integrating live IoT feeds for dynamic monitoring, and adapting thresholds to node-specific or regional standards. Testing across diverse network topologies, climates, and noise levels will further validate generalizability. Additional research could explore hybrid models (e.g., LSTM with GBT) or cloud deployment strategies (e.g., Docker, Kubernetes) to enhance operational readiness, solidifying this approach as a cornerstone of smart infrastructure resilience.

References

[1] A. de la Fuente, C. Meruane, and V. Meruane, "Ensemble weather-runoff forecasting models for reliable flood early warning systems," *Progress in Disaster Science*, p. 100420, 2025.

[2] P. Muñoz, J. Orellana-Alvear, J. Bendix, J. Feyen, and R. Célleri, "Flood early warning systems using machine learning techniques: The case of the tomebamba catchment at the southern andes of ecuador," *Hydrology*, vol. 8, no. 4, 2021.

[3] F. Piadeh, K. Behzadian, A. S. Chen, Z. Kapelan, J. P. Rizzuto, and L. C. Campos, "Enhancing urban flood forecasting in drainage systems using dynamic ensemble-based data mining," *Water Research*, vol. 247, p. 120791, 2023.

[4] A. Faruq, S. F. M. Hussein, A. Marto, and S. S. Abdullah, "Flood river water level forecasting using ensemble machine learning for early warning systems," *IOP Conference Series: Earth and Environmental Science*, vol. 1091, p. 012041, nov 2022.

[5] F. Sseguya and K.-S. Jun, "Deep learning ensemble for flood probability analysis," *Water*, vol. 16, no. 21, 2024.