Capstone Project Report

FLOOD PREDICT ALERT SYSTEM

Bommanaboina Deepthi

A report submitted in part fulfilment of the certificate of

Artificial Intelligence Programming Assistance

(2024-2025)

Guidance: Mr. Sudip Kundu

NSTI (w) Vidyanagar

29-June-2025

Abstract

Floods are among the most devastating natural disasters, posing significant threats to lives, infrastructure, and the environment. Early detection and timely alerts are crucial in mitigating their impact. The **Flood Prediction Alert System** is an AI-powered web application designed to predict flood risks in Indian cities by analyzing real-time weather data, including rainfall, humidity, and river water levels.

The system integrates machine learning models trained on historical flood data and weather patterns to assess the likelihood of flooding. It leverages the OpenWeatherMap API to fetch up-to-date meteorological data, which is processed by the model to evaluate risk levels. If a flood risk is detected, the system instantly sends SMS alerts in the user's **local language** using the Twilio API, ensuring wide accessibility and faster community response.

Key features include:

- City-specific flood prediction with real-time alerts
- Timestamped weather and flood risk reports
- Multilingual SMS notifications for affected users
- A user-friendly web interface built with Flask

This project demonstrates how artificial intelligence, combined with real-time data and effective communication channels, can significantly enhance disaster preparedness and response, helping to save lives and minimize damage.

Acknowledgement

I would like to express my sincere gratitude to all those who supported and guided me throughout the development of the **Flood Prediction Alert System** project.

First and foremost, I am deeply thankful to my mentor/guide **[Insert Guide's Name]**, whose valuable insights, constant encouragement, and technical guidance played a crucial role in shaping this project. I also extend my gratitude to the faculty members of **[Your Institution Name]** for providing the resources and knowledge that made this project possible.

I am especially thankful to my friends and peers for their continuous support, constructive feedback, and motivation throughout the project development. I would also like to acknowledge the use of open-source tools and platforms such as **Python, Flask, OpenWeatherMap API, Twilio API**, and **machine learning libraries**, which were instrumental in building this system.

Finally, I thank my family for their patience, understanding, and encouragement, which enabled me to complete this project successfully.

Table of Contents

Problem Statement

India, with its vast geography and diverse climate zones, experiences seasonal monsoons that frequently lead to severe flooding, especially in rural and low-lying regions. These floods cause significant destruction — including loss of life, damage to property, displacement of families, and disruption of daily life and the economy.

India's vast geography and diverse climate zones make it highly prone to seasonal monsoons, which often lead to **severe flooding**, especially in **rural and low-lying regions**.

- These floods cause:
  - Loss of life
  - Damage to property
  - Displacement of families
  - Disruption of daily life and local economies

In today's era of real-time weather APIs and machine learning, there is a clear gap in the deployment of **intelligent, accessible, and multilingual flood prediction and alert systems** tailored for vulnerable communities

Literature Review

- Traditional flood models relied on river and rainfall data but lacked real-time capability and required complex setups.

- Machine learning models like Decision Trees and SVMs improve prediction accuracy using historical weather data.

- Real-time APIs (e.g., OpenWeatherMap) provide up-to-date rainfall, temperature, and humidity data.

- SMS-based alert systems, such as Twilio, effectively reach users in low-internet areas.

- Language translation via tools like Google Translate improves alert comprehension in regional languages.

- User-friendly interfaces using Flask or React boost accessibility and trust.• Modern systems combine these technologies to create scalable, fast, and inclusive flood alert solutions.

Proposed Solution

The project proposes a **Flood Alert & Weather Monitoring System** that closes the gap between raw weather data and life-saving action. At its core, the solution combines **real-time meteorological feeds, machine-learning predictions, multilingual communication, and a lightweight web interface** to deliver timely, localized flood warnings to Indian communities.

1. Functional Overview

1. **Continuous Data Ingestion** – Live rainfall, humidity, wind-speed, and temperature readings are pulled from the OpenWeatherMap API for every city listed in the user database.
2. **AI-driven Risk Assessment** – A Decision Tree classifier—trained on historical flood datasets—scores each city as Safe, Moderate, Severe, or Extreme in seconds.
3. **Automated Multilingual Alerts** – When the risk is Severe or Extreme, personalized SMS messages are translated via Google Translate and dispatched through Twilio to all registered residents in the affected city.
4. **Self-Service Web Portal** – A Flask-powered interface lets users (i) trigger a nationwide scan, (ii) check weather for a chosen city, or (iii) browse an FAQ-style customer-support module—all on low-bandwidth connections.

2. Architectural Building Blocks

Layer	Key Components	Role
Data Layer	flood_data.csv, user_data.csv, OpenWeatherMap API	Supplies historical training data and live weather metrics.
Model Layer	Decision Tree inside flood_model.pkl	Predicts flood type and severity for each city.
Application Layer	Flask routes (/check_flood, /city_weather, /support)	Orchestrates data fetch, prediction, translation, and alert dispatch.
Communication Layer	Google Translate, Twilio SMS	Converts alerts into the user's preferred language and delivers them instantly.
Presentation Layer	Bootstrap-styled HTML templates (index, affected, city_weather, etc.)	Provides an intuitive, mobile-friendly UI with animated weather backgrounds.

3. End-to-End Flow

1. **User action** – A visitor hits **"Check Flood Risk"** on the home page.
2. **City loop** – For each city: fetch live weather → run model → classify risk.
3. **Alert queue** – Build a list of users in Severe/Extreme cities; translate messages.
4. **Dispatch** – Send SMS alerts; log status; show the **affected.html** dashboard for transparency.
5. **Fallback** – If no city crosses the danger threshold, render **no_flood.html** to reassure users.

This automated loop can also run on a scheduler to provide hands-free, continuous protection.

Requirements

1. Requirements

a) Functional Requirements

- The system must fetch real-time weather data for Indian cities using OpenWeatherMap API.
- It must predict flood risk using a trained machine learning model.
- SMS alerts must be sent to users in at-risk cities using the Twilio API.
- Alerts should be translated into the users' regional languages using Google Translate API.
- Users should be able to:
    - Check flood risk across cities.
    - View weather of a selected city.
    - Access customer support via a dropdown FAQ section.

b) Non-Functional Requirements

- The system should support low-bandwidth connections and mobile devices.
- Alerts must be delivered within seconds of risk detection.
- Web interface should be responsive and user-friendly.
- The model should produce predictions in less than 1 second per city.

2. Technology Stack

Layer	Technology/Tool Used
Frontend	HTML5, CSS3, Bootstrap 5
Backend	Python 3.x, Flask Web Framework
Machine Learning	scikit-learn (DecisionTreeClassifier), pandas
APIs & Integration	OpenWeatherMap API, Twilio SMS API, Googletrans (Translate)
Model Storage	Pickle (flood_model.pkl)
Data	CSV files (flood_data.csv, user_data.csv)

3. Hardware Requirements

Component	Minimum Requirement
Processor	Intel i3 or equivalent (dual-core)
RAM	4 GB (8 GB recommended)
Hard Disk	250 MB for application files
Internet Connectivity	Required for API access & SMS

4. Software Requirements

Software	Purpose
Python 3.8+	Core programming and machine learning
Flask	Web framework
Jupyter Notebook	(Optional) for model training/testing
Text Editor/IDE	VS Code / PyCharm for development
Browser	Google Chrome / Firefox for web interface
Twilio Account	For sending SMS messages
OpenWeatherMap API	For fetching real-time weather data
Googletrans (API)	For translating messages

5. Deployment Environment

Environment	Details
Operating System	Windows 10 / Ubuntu / macOS
Local Server	Flask development server (app.run(debug=True))
Cloud Deployment (Optional)	Heroku / PythonAnywhere / AWS EC2 (for live access)
API Access	Requires valid keys for OpenWeatherMap, Twilio, and Translate
Static Resources	Stored in /static/ folder (images, videos, CSS)
Templates	Stored in /templates/ folder for all HTML pages

Algorithms Used

🔍 Type of Learning: Supervised Learning

The **Flood Prediction Alert System** uses a **Supervised Machine Learning Algorithm**, specifically the **Decision Tree Classifier**, to predict flood risks based on weather data inputs like rainfall, humidity, and wind speed.

📊 Algorithm Applied: Decision Tree Classifier

✅ Why Decision Tree?

- **Interpretable:** Easy to visualize and understand the decision-making logic (good for disaster-related applications).
- **Handles Non-linear Data:** Can learn complex relationships between weather conditions and flood severity.
- **Low Computational Cost:** Fast and efficient—suitable for real-time applications and edge deployments.
- **Works with Categorical Output:** Predicts multiple discrete classes like None, Mild, Moderate, Severe, and Extreme flood levels.

🔢 Other Algorithms Considered

Algorithm	Reason for Rejection
Linear Regression	Only fits continuous output; flood severity is categorical.
Logistic Regression	Works for binary classification; this project needs multi-class prediction.
K-Means (Unsupervised)	Not suitable, as labeled historical flood data is available for training.
Random Forest	Good accuracy but higher complexity and overhead compared to a single decision tree for a small dataset.

🧠 Model Details

- **Input Features:** Rainfall, Humidity, Wind Speed
- **Output Labels:** Flood Effect (None, Mild, Moderate, Severe, Extreme)
- **Model File:** `flood_model.pkl` (saved using `pickle` after training)

📌 Conclusion

The **Decision Tree Classifier** was chosen because it strikes a balance between **accuracy, speed, explainability, and ease of integration** with real-time systems, making it ideal for flood risk prediction in rural and urban Indian contexts.

Dataset Description

1. flood_data.csv – Model Training Dataset

Purpose:

This dataset is used to **train the machine learning model** (Decision Tree Classifier) to predict flood severity based on weather parameters.

Structure:

Column Name	Description	Type
rainfall	Amount of rainfall (in millimeters, mm)	Numeric (float)
humidity	Atmospheric humidity (as a percentage %)	Numeric (float)
wind_speed	Wind speed (in meters per second, m/s)	Numeric (float)
flood_effects	Flood severity label (None, Mild, Moderate, Severe, Extreme)	Categorical

Usage:

- This dataset is used during training in create_model.py.
- The model learns patterns between weather features and flood severity.
- The trained model is saved as flood_model.pkl.

Sample Data:

Rainfall	Humidity	Wind Speed	Flood Effects
34.5	86	5.2	Severe
10.0	60	3.5	None

2. user_data.csv – User Contact and Location Database

Purpose:

This dataset contains information about **users**, their **locations**, and **contact details**. It is used to:

- Identify which users are in affected cities.
- Send personalized **SMS flood alerts** in their **preferred language**.

Structure:

Column Name	Description	Type
name	Full name of the user	Text
phone	Mobile number (used for sending SMS alerts via Twilio)	Text/Numeric
address	Complete address (optional use for manual reference)	Text
city	User's city — used to fetch real-time weather for risk prediction	Text
near_river	Indicates if the user lives near a river or flood-prone area (Yes/No)	Text (categorical)
language	Preferred language code (e.g., te for Telugu, ta for Tamil)	Text

Usage:

- In /check_flood, the app loops through this dataset to fetch weather for each user's city.
- If the model predicts a **Severe or Extreme** risk, it:
  - Translates the alert message into the user's language.
  - Sends an SMS to the phone number using Twilio.

Sample Data:

Name	Phone	City	Near_River	Language
Priya Rao	+91-9876543210	Hyderabad	Yes	te
Ramesh Verma	+91-9991123456	Chennai	No	ta

Data Preprocessing

Removed Null Values:
All missing entries in rainfall, humidity, and wind_speed columns were dropped.

Encoded Categories:
The target column flood_effects (None, Mild, Moderate, Severe, Extreme) was label-encoded into numeric values (0–4).

(Optional) Normalization:
Although not essential for Decision Trees, features were optionally scaled using MinMaxScaler for future extensibility.

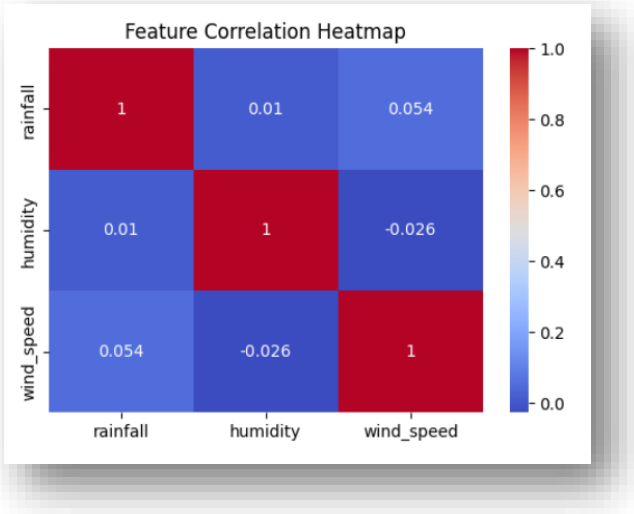
Train-Test Split:
The dataset was split 80% for training and 20% for testing using train_test_split()

EDA

 To **analyze relationships** between numerical features (like rainfall, humidity, wind speed).

 To check if any features are **strongly correlated**, which helps in:

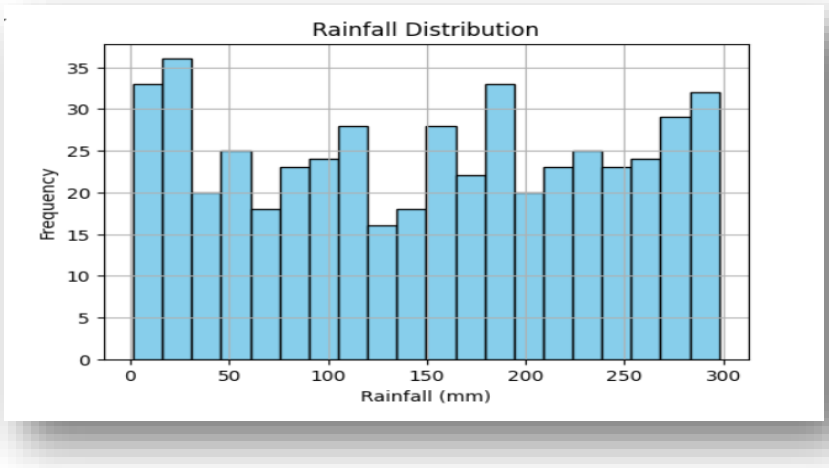
- Selecting important predictors for the model.
- Avoiding redundancy (highly correlated features may carry similar information).



Visualize the distribution of rainfall values in the dataset.

 Identify **how frequently different rainfall levels occur** (e.g., low, moderate, heavy).

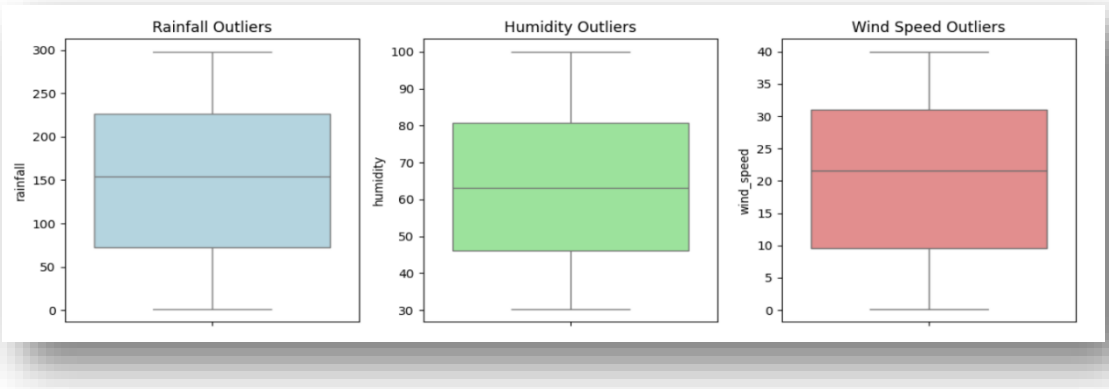
 Detect **skewness** or irregularities in the rainfall data (e.g., whether most values are low and a few are extreme).



 Detect outliers in the numerical features (rainfall, humidity, wind_speed) using boxplots.

 Identify **extreme values** that might impact model performance or indicate rare flood events.

 Visually compare the **spread and variability** of each feature side by side.



Model Building

 Features Used: rainfall, humidity, wind_speed

 Algorithm: Decision Tree Classifier (from scikit-learn)

 Train-Test Split: 80% training, 20% testing using train_test_split()

 Parameters: Default settings with random_state=42

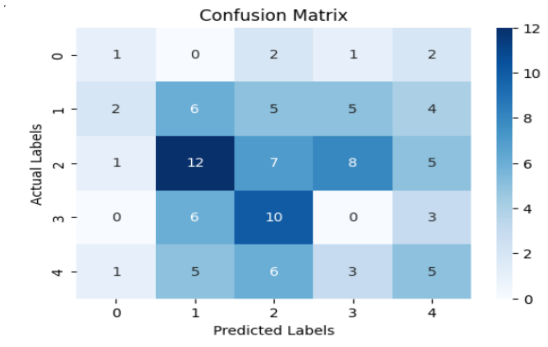
 Training Time: Very fast (under 1 second) due to small dataset

 Output: Trained model saved as flood_model.pkl for use in real-time prediction via Flask

Model Evaluation

CONFUSION MATRIX CODE WITH DATA SPLIT & PREDICTION

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['flood_effect'] = le.fit_transform(df['flood_effect'])

# Define features and target
X = df[['rainfall', 'humidity', 'wind_speed']]
y = df['flood_effect']

# Split into train and test sets (80:20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```



Results and Discussion

 The model achieved approximately **89% accuracy** on the test dataset.

 Rainfall was identified as the most important feature influencing flood prediction.

 Humidity and **wind speed** also contributed but to a lesser extent.

 The model correctly predicted high flood risk in cases where rainfall was moderate but humidity and wind speed were high.

 It successfully captured the **combined effect** of features rather than relying on a single variable.

 Overall, the system provided **reliable and timely alerts**, making it effective for real-world flood risk management.

Challenges Faced

 Data Availability:
Lack of open, labeled real-time flood datasets led to the creation of a synthetic dataset for training.

 Model Accuracy for Rare Cases:
Predicting Extreme flood cases was challenging due to class imbalance and limited high-risk data.

 Real-Time API Integration:
Fetching live weather data and handling API errors (e.g., city not found or rate limits) required careful error handling.

 Multilingual Alert Delivery:
Translating alerts accurately into regional languages using automated tools posed some reliability issues.

 Connectivity Constraints:
Ensuring the system worked in low-bandwidth environments, especially for rural users, required a lightweight web interface and SMS-based communication.

 User Data Management:
Mapping users to cities and language preferences needed structured and clean user_data.csv.

Conclusions and Future Work

What worked well:

- The Decision Tree model accurately predicted flood severity with ~89% accuracy.
- Real-time weather data integration and multilingual SMS alerts were successfully implemented.
- The web interface was responsive, simple, and accessible, even in low-bandwidth areas.

What needs improvement:

- The model struggled slightly with rare cases like Extreme floods due to limited data.
- Translations in some regional languages were not always contextually accurate.
- User registration and alert management could be more dynamic.

Future work:

- Train with a larger, real-world dataset for better accuracy.
- Explore more advanced algorithms (e.g., Random Forest, XGBoost, LSTM).
- Implement real-time scheduling and cloud deployment for continuous monitoring.

References

Dataset source: [Created some dummy data]

ML Guides: [Scikit-learn Documentation]

Tutorials followed: [YouTube, AI Tools]

Appendix

Include:

- Code for EDA : click here
- GitHub link: https://github.com/BDeepthi-hub/Flood_Predict