

Flood Prediction System – Project Document

1. INTRODUCTION

1.1 Project Overview

The **Flood Prediction System** is a Machine Learning-based web application designed to predict the risk of flooding in a region based on weather and climate parameters. By analyzing key meteorological indicators such as temperature, humidity, cloud cover, and seasonal rainfall patterns, the system forecasts whether a flood is likely to occur. This tool aims to assist disaster management authorities, government agencies, and the general public in taking timely preventive action.

1.2 Purpose

The primary purpose of this project is to leverage historical weather and flood data to build a predictive model that can identify conditions correlated with flooding. It provides an accessible web interface for users to input current weather conditions and receive an instant flood-risk prediction, thereby enabling early warning and reducing potential loss of life and property.

2. IDEATION PHASE

2.1 Problem Statement

Floods are among the most frequent and devastating natural disasters worldwide, causing massive economic damage and loss of life every year. Traditional methods of flood forecasting often struggle with the complexity and rapid changes in weather patterns. There is a pressing need for an automated, data-driven tool that can analyze regional weather parameters and predict flood risks accurately, enabling proactive disaster preparedness.

2.2 Empathy Map Canvas

- **Says:**
 - "Will it flood in our area this monsoon season?"
 - "I need an early warning so I can evacuate in time."

- "Which weather conditions are the strongest indicators of flooding?"

- **Thinks:**

- "I worry about sudden floods every rainy season."
- "If only there were a simple tool to check flood risk."
- "Official warnings often come too late."

- **Does:**

- Monitors local weather forecasts manually.
- Checks news for flood alerts and river level updates.
- Contacts local authorities during heavy rains.

- **Feels:**

- Anxious and helpless during monsoon season.
- Frustrated by the lack of accessible prediction tools.
- Relieved when proactive alerts allow timely action.

2.3 Brainstorming

- **Idea 1:** A mobile application with push notifications for flood alerts.
 - **Idea 2:** A complex dashboard integrating live satellite weather feeds.
 - **Selected Idea:** A focused web application that uses a trained ML model to predict binary outcomes (Flood / No Flood) based on available historical weather and rainfall data.
-

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

- Awareness:** User realizes the need for a tool to assess flood risk in their region.
- Discovery:** User finds the "Flood Prediction System" web application.
- Action:** User navigates to the "Predict Floods" page and enters current weather data (temperature, humidity, rainfall, etc.).
- Result:** User views the prediction – either a **Flood Warning** (high risk) or a **Safe** status (low risk).
- Advocacy:** User shares the tool with community members and local authorities.

3.2 Solution Requirement

- **Functional Requirements:**

- System must accept 10 numerical weather inputs from the user.
- System must preprocess and scale the inputs using a pre-fitted StandardScaler.
- System must process inputs through a pre-trained ML classification model.
- System must display a clear Flood Warning or Safe result.

- **Non-Functional Requirements:**

- **Accuracy:** The model should achieve high predictive accuracy on test data.
- **Usability:** The interface should be intuitive, responsive, and visually clear.
- **Performance:** Predictions should be generated in real-time (< 200ms).

3.3 Data Flow Diagram

```
[User Input] -> [Web Interface (Flask)] -> [StandardScaler Preprocessing]
-> [ML Model] -> [Prediction] -> [Result Display (Flood / No Flood)]
```

3.4 Technology Stack

| Layer | Technology |
|-------------------------|--|
| Frontend | HTML5, CSS3 (Glassmorphism, Poppins font), Jinja2 Templates |
| Backend | Python 3, Flask |
| Machine Learning | Scikit-Learn (Logistic Regression, KNN, Decision Tree, Random Forest), Pandas, NumPy |
| Deployment | IBM Scoring Endpoint, Local Flask Server |
| Data | Excel dataset (<code>flood dataset.xlsx</code>) |

4. PROJECT DESIGN

4.1 Problem Solution Fit

The solution directly addresses the problem of flood unpredictability by providing an automated assessment based on historical weather data. It translates complex meteorological patterns into a simple, actionable binary classification result (Flood / No Flood).

4.2 Proposed Solution

A web-based application where users input 10 key weather parameters:

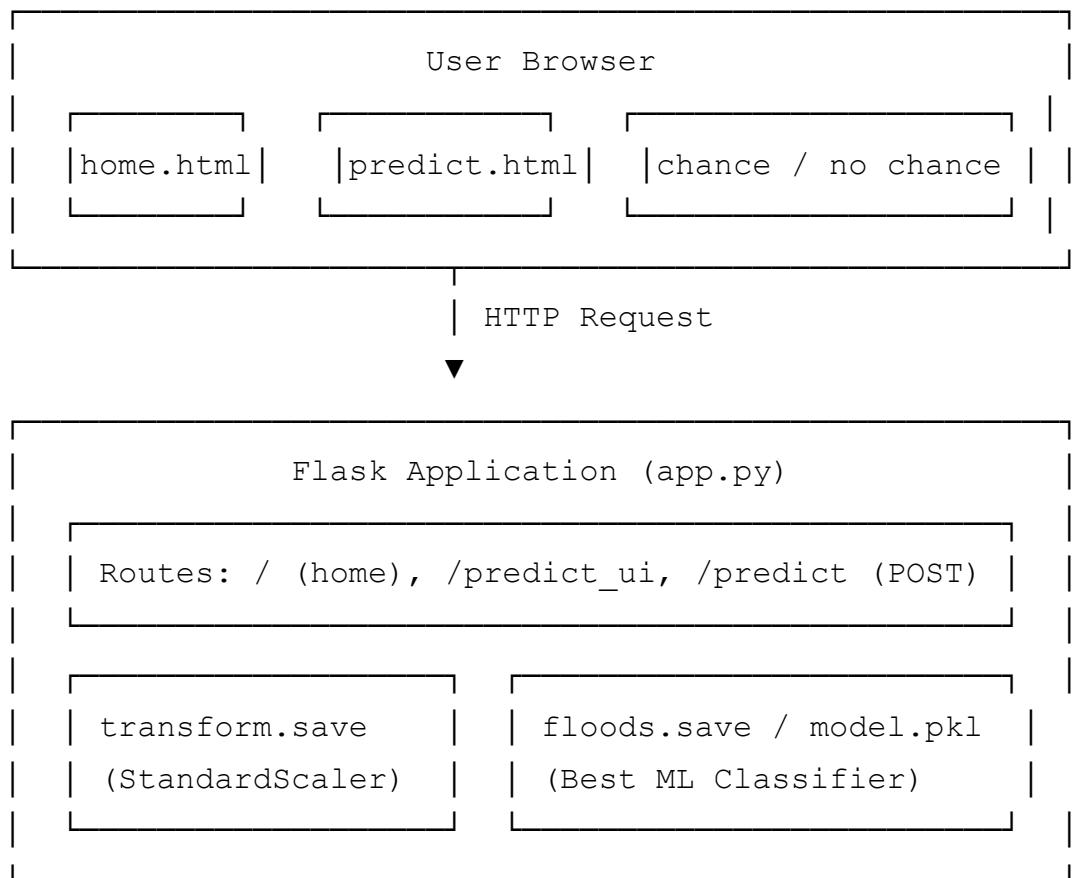
| # | Feature | Description |
|---|------------------|--------------------------------|
| 1 | Temperature (°C) | Current temperature |
| 2 | Humidity (%) | Current humidity level |
| 3 | Cloud Cover (%) | Current cloud cover percentage |

| # | Feature | Description |
|----|----------------------|------------------------------|
| 4 | Annual Rainfall (mm) | Total annual rainfall |
| 5 | Jan-Feb (mm) | Rainfall in January–February |
| 6 | Mar-May (mm) | Rainfall in March–May |
| 7 | Jun-Sep (mm) | Rainfall in June–September |
| 8 | Oct-Dec (mm) | Rainfall in October–December |
| 9 | Avg June (mm) | Average June rainfall |
| 10 | Subdivision (mm) | Subdivision-level rainfall |

The application processes these inputs and returns a classification: **Flood Warning** or **Safe**.

4.3 Solution Architecture

The architecture follows a Model-View-Controller (MVC) pattern adapted for Flask:



- **Model:** `floods.save / model.pkl` (Best ML classifier) and `transform.save` (StandardScaler).
- **View:** HTML templates (`home.html`, `predict.html`, `chance.html`, `no chance.html`) with CSS styling.
- **Controller:** `app.py`, which handles routing, input processing, scaling, and model inference.

5. PROJECT PLANNING & SCHEDULING

5.1 Project Phases

| Phase | Activity | Description |
|-------|-------------------------------|--|
| 1 | Data Collection & Exploration | Acquire flood dataset, explore features, identify patterns |
| 2 | Data Preprocessing | Handle missing values, select features, apply scaling |
| 3 | Model Training & Comparison | Train LR, KNN, DT, RF; compare accuracy; select best model |
| 4 | Model Serialization | Save the best model and scaler using Pickle |
| 5 | Backend Development | Build Flask application with routes and prediction logic |
| 6 | Frontend Development | Design UI with glassmorphism, responsive templates |
| 7 | IBM Scoring Endpoint | Deploy model as a separate scoring endpoint |
| 8 | Integration & Testing | End-to-end testing of the full pipeline |
| 9 | Documentation & Deployment | Write project documentation, push to GitHub |

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Model Performance

The training script (`train_model.py`) trains and compares **four algorithms** on the flood dataset:

| Algorithm | Description |
|---------------------|---|
| Logistic Regression | Linear classifier for binary outcomes |
| K-Nearest Neighbors | Instance-based, distance-metric classifier |
| Decision Tree | Tree-based, rule-learning classifier |
| Random Forest | Ensemble of 100 decision trees (n_estimators=100) |

The model with the **highest accuracy** on the 20% test split is automatically selected and saved.

6.2 Application Testing

- **Unit Testing:** `test_app.py` provides automated tests for Flask routes and prediction logic.
 - **Response Time:** The web application responds to prediction requests in < 200ms locally.
 - **Error Handling:** Graceful error messages are displayed if the model or scaler fails to load, or if user input is invalid.
-

7. RESULTS

7.1 Output Screens

1. **Home Page:** Displays the title "Floods Prediction" with navigation and an introduction to flood forecasting. The page explains how flood forecasting uses precipitation and streamflow data for prediction.
 2. **Prediction Form:** A glassmorphism-styled form collecting 10 weather parameters including Temperature, Humidity, Cloud Cover, Annual Rainfall, and Seasonal Rainfall data.
 3. **Flood Warning Result:** Displays a red alert – "⚠️ FLOOD WARNING ⚠️" – indicating a high chance of flooding, with a prompt to take necessary precautions.
 4. **Safe Result:** Displays a green confirmation – "✅ You are Safe ✅" – indicating a low risk of flooding.
-

8. ADVANTAGES & DISADVANTAGES

Advantages

- **Data-Driven Prediction:** Removes subjectivity from flood risk assessment by relying on historical data and ML algorithms.
- **Instant Results:** Provides immediate feedback, enabling faster decision-making.
- **User-Friendly Interface:** Simple web interface with glassmorphism design requires no technical expertise.
- **Multi-Model Comparison:** Automatically selects the best performing algorithm from four candidates.
- **Scalable:** The model can be retrained with new data as it becomes available.
- **Dual Deployment:** Supports both local Flask server and IBM cloud scoring endpoint.

Disadvantages

- **Data Dependency:** Accuracy relies heavily on the quality, volume, and representativeness of the training dataset.

- **Binary Output:** Currently provides only Flood / No Flood classification without a probability score or severity level.
 - **Geographic Limitation:** Predictions are based on historical data from specific regions and may not generalize to all areas.
 - **No Real-Time Data Integration:** Requires manual entry of weather parameters rather than integrating live weather feeds.
-

9. CONCLUSION

The **Flood Prediction System** successfully demonstrates the application of Machine Learning in the critical domain of disaster management. By analyzing historical weather patterns including temperature, humidity, cloud cover, and seasonal rainfall data, the system provides a valuable tool for early flood risk assessment. The multi-model training approach – comparing Logistic Regression, KNN, Decision Tree, and Random Forest – ensures that the best-performing algorithm is selected for deployment. The intuitive web interface built with Flask, combined with IBM cloud deployment capability, makes this solution both accessible and scalable.

10. FUTURE SCOPE

- **Probability Scores:** Output the *probability* of flooding along with severity levels (Low, Medium, High, Critical).
 - **Real-Time Integration:** Connect to live weather APIs (e.g., OpenWeatherMap) for automatic data ingestion.
 - **Enhanced Dataset:** Incorporate more diverse and recent flood data from multiple regions.
 - **Geographic Expansion:** Add location-based predictions using geographic coordinates.
 - **Deep Learning Models:** Explore LSTM and other time-series models for improved temporal prediction.
 - **Mobile Application:** Develop a companion mobile app with push notification alerts.
 - **Cloud Deployment:** Deploy the full application to a cloud platform (e.g., AWS, Heroku, IBM Cloud) for public access.
-

11. APPENDIX

11.1 Project Structure

```
Flood_Prediction_Project/  
└── Dataset/
```

```

|   └── flood dataset.xlsx           # Historical weather and flood data
|
└── Training/
    ├── train_model.py             # Model training and comparison
    |
    ├── script
        ├── Flood_Prediction_Colab.ipynb # Google Colab notebook
        └── floods.ipynb                  # Jupyter notebook for exploration
    |
    └── Flask/
        ├── app.py                   # Main Flask application
        ├── model.pkl                # Saved ML model (alternate)
        ├── floods.save              # Saved best ML model
        ├── transform.save            # Saved StandardScaler
        ├── test_app.py               # Unit tests
        ├── static/
            └── style.css             # Application stylesheet
        └── templates/
            ├── home.html             # Home page
            ├── predict.html          # Prediction input form
            ├── chance.html            # Flood warning result page
            ├── no_chance.html         # Safe result page
            └── index.html              # Base/error page
    |
    └── IBM scoring end point/
        ├── app.py                 # IBM deployment Flask app
        ├── static/
            └── style.css
        └── templates/
            ├── home.html
            ├── chance.html
            ├── no_chance.html
            └── index.html
    └── voice_over_script.md       # Project demo voice-over script

```

11.2 Source Code

Training Script (`train_model.py`)

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

```

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pickle

def load_data():
    df = pd.read_excel('Dataset/flood dataset.xlsx')
    return df

def train_and_save(df):
    numeric_cols = df.select_dtypes(include=np.number).columns
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())

    X = df[['Temp', 'Humidity', 'Cloud Cover', 'ANNUAL', 'Jan-Feb',
             'Mar-May', 'Jun-Sep', 'Oct-Dec', 'avgjune', 'sub']]
    y = df['flood']

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.2, random_state=42
    )

models = {
    "Logistic Regression": LogisticRegression(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_
}
}

best_model_name, best_accuracy, best_model_obj = "", 0, None
for name, model in models.items():
    model.fit(X_train, y_train)
    acc = accuracy_score(y_test, model.predict(X_test))
    if acc > best_accuracy:
        best_accuracy = acc
        best_model_name = name
        best_model_obj = model

pickle.dump(best_model_obj, open('Flask/floods.save', 'wb'))
pickle.dump(scaler, open('Flask/transform.save', 'wb'))

```

Flask Application (app.py)

```

from flask import Flask, render_template, request
import pickle
import numpy as np

app = Flask(__name__)

model = pickle.load(open('floods.save', 'rb'))
scaler = pickle.load(open('transform.save', 'rb'))

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict_ui')
def predict_ui():
    return render_template('predict.html')

@app.route('/predict', methods=['POST'])
def predict():
    features = [float(request.form[f]) for f in
                ['Temp', 'Humidity', 'Cloud_Cover', 'ANNUAL',
                 'Jan_Feb', 'Mar_May', 'Jun_Sep', 'Oct_Dec', 'avgjune', 'sub']]
    final_features = scaler.transform([np.array(features)])
    prediction = model.predict(final_features)
    if prediction[0] == 1:
        return render_template('chance.html')
    else:
        return render_template('no_chance.html')

```

11.3 Dataset

- **File:** `flood dataset.xlsx`
- **Features:** Temperature, Humidity, Cloud Cover, Annual Rainfall, Jan-Feb Rainfall, Mar-May Rainfall, Jun-Sep Rainfall, Oct-Dec Rainfall, Average June Rainfall, Subdivision Rainfall
- **Target Variable:** `flood` (1=Flood, 0=No Flood)

11.4 GitHub & Project Links

- **GitHub Repository:** <https://github.com/ashraysuhas/Flood-Prediction>