Weather Prediction Using Machine Learning

Task 1

Team I3S

Content

1. Introduction

1.1 Overview

Weather prediction plays a crucial role in various sectors, including agriculture, transportation, and disaster management. Traditional weather forecasting methods rely on physics-based models and satellite data, which, while useful, often lack the precision needed for hyper-local predictions. Machine Learning (ML) offers a data-driven approach that can improve forecast accuracy by learning patterns from historical weather data.

This report outlines a step-by-step methodology for building an ML model to predict rainfall based on historical weather observations. The dataset consists of daily weather data for 300 days, including key meteorological parameters such as **average temperature, humidity, and wind speed**. The target variable, **rain_or_not**, indicates whether it rained (1) or not (0) on a given day.

Beyond model development, the report also proposes a real-time weather prediction system using **IoT (Internet of Things) devices and sensors**. These devices collect weather data at one-minute intervals, enabling continuous monitoring. Since IoT sensors can sometimes malfunction, the system is designed with fault-tolerant mechanisms to handle missing or inconsistent data.

The implementation of this ML-based weather prediction system can help **farmers** plan irrigation schedules, planting seasons, and harvesting times more efficiently. By leveraging past weather data and real-time sensor readings, the system aims to improve decision-making and mitigate risks associated with unpredictable weather conditions.

1.2 Objectives

The primary objective of this project is to **develop an accurate and reliable machine learning model** for predicting rainfall. To achieve this, we define the following goals:

1. **Data Preprocessing:**
    - Handle missing values, incorrect entries, and inconsistencies in the dataset.

    - Convert categorical variables (if any) into numerical representations.
    - Standardize or normalize numerical features to improve model performance.
2. **Exploratory Data Analysis (EDA):**
    - Analyze trends and relationships between features.
    - Visualize key insights using graphs and statistical methods.
    - Identify patterns that influence rainfall occurrences.
3. **Model Training and Evaluation:**
    - Train multiple machine learning models such as **Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting**.
    - Evaluate the models based on performance metrics like **accuracy, precision, recall, and F1-score**.
4. **Hyperparameter Tuning and Optimization:**
    - Fine-tune model parameters to achieve the best accuracy.
    - Perform feature selection to remove unnecessary data and improve efficiency.
5. **Rainfall Prediction for Future 21 Days:**
    - Use the trained model to predict whether it will rain for the next 21 days.
    - Provide a probability score indicating the likelihood of rain.
6. **System Design for Real-Time Weather Prediction:**
    - Develop a framework where IoT sensors continuously collect weather data.
    - Implement fault-tolerant mechanisms to handle sensor malfunctions.
    - Create an architecture diagram showing how data flows from sensors to the end-user prediction system.

2. Data Preprocessing

2.1 Dataset Description

The dataset **weather_data.csv** contains **300 days** of historical weather observations. It includes various meteorological attributes that influence rainfall patterns. Understanding the dataset structure is crucial for data cleaning and feature engineering. Below are the key attributes included in the dataset:

1. **Date:**
    - Represents the specific day of observation.
    - Helps in identifying time-based trends and seasonal patterns.
    - Format: **YYYY-MM-DD** (e.g., 2024-03-09).
2. **Avg Temperature (°C):**
    - The **mean daily temperature** recorded for that particular day.
    - Important because temperature influences cloud formation, evaporation, and precipitation.
    - Values typically range between **low (cold days) and high (hot days)**.
    - Units: **Degrees Celsius (°C)**.
3. **Humidity (%):**
    - Represents the amount of **moisture** present in the air.
    - High humidity increases the likelihood of precipitation.
    - Measured as a **percentage (%)**, where **100%** means full air saturation, leading to condensation and rain.
4. **Avg Wind Speed (km/h):**
    - The **average speed of the wind** on a given day.
    - Wind affects **cloud movement, moisture distribution, and storm formation**.
    - Measured in **kilometers per hour (km/h)**.
    - Higher wind speeds may **disperse clouds** and reduce rainfall probability.
5. **Rain or Not (Binary Label):**
    - The **target variable (dependent variable)** that we want to predict.

- Binary classification:
  - **1:** Indicates that it rained on that day.
  - **0:** Indicates that it did not rain.
- This variable will be used as the **output label** for training the machine learning model.

2.2 Handling Missing Values

[1]:
```
#Loading the dataset
import pandas as pd

df = pd.read_csv("C:/weather_data.csv")

print(df.head())
```

```
date avg_temperature humidity avg_wind_speed rain_or_not \ •••
```

[3]:
```
print(df['rain_or_not'].unique())
```

```
['Rain' 'No Rain'] •••
```

[5]:
```
df['rain_or_not'] = df['rain_or_not'].map({'Rain': 1, 'No Rain': 0})  # Convert categorical values
df['rain_or_not'] = df['rain_or_not'].fillna(0).astype(int)  # Ensure integer type

print(df.head())
```

```
date avg_temperature humidity avg_wind_speed rain_or_not \ •••
```

[7]:
```
#Convert the 'date' Column to Date Format
df['date'] = pd.to_datetime(df['date'], errors='coerce')

print(df.head())
```

```
date avg_temperature humidity avg_wind_speed rain_or_not \ •••
```

[9]:
```
#Fill Missing Values
df.fillna(df.mean(numeric_only=True), inplace=True)
```

[13]:
```
#Check Data Again
print(df.info())  # Shows the structure of the dataset
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'> •••
```

To clean the dataset:

- Convert date to datetime format.
- Replace missing numerical values with the column mean.
- Convert categorical labels (Rain / No Rain) to binary values (1 / 0).

3. Exploratory Data Analysis (EDA)

3.1 Descriptive Statistics

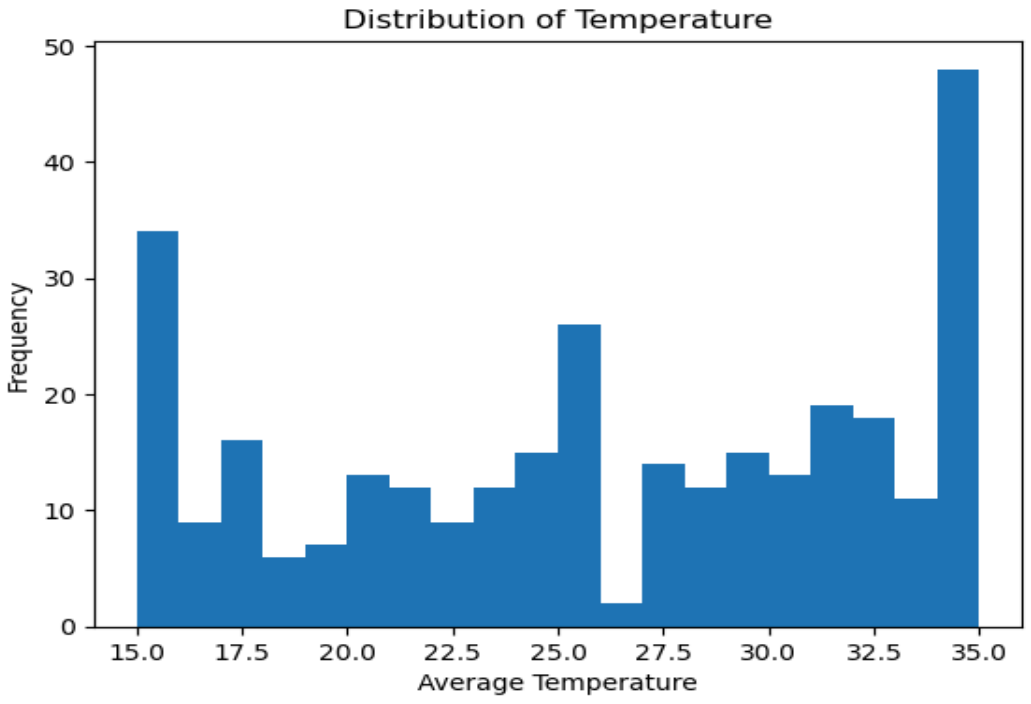
```
[15]: #Exploratory Data Analysis (EDA)
      #Check Basic Statistics
      print(df.describe())
```

Basic statistics provide insight into the dataset's distribution.

3.2 Data Visualization

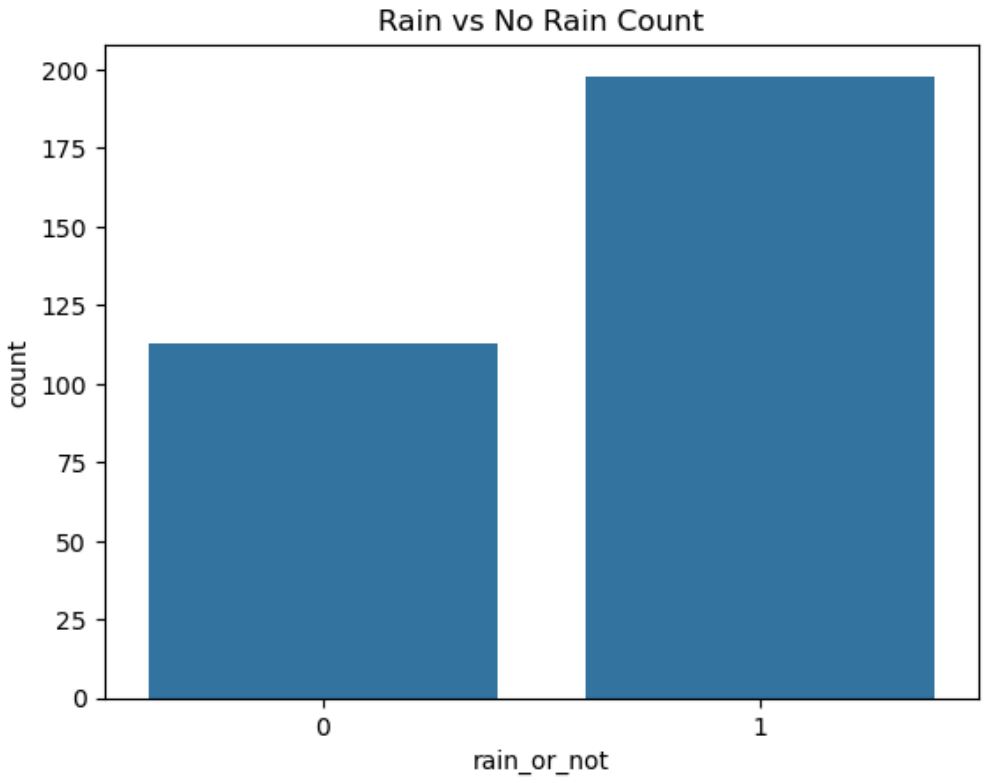
- **Temperature Distribution:**

```
[17]: #Visualize Data
      #Histogram of Temperature
      import matplotlib.pyplot as plt

      plt.hist(df['avg_temperature'], bins=20)
      plt.xlabel('Average Temperature')
      plt.ylabel('Frequency')
      plt.title('Distribution of Temperature')
      plt.show()
```



- **Rain vs No Rain Count:**

[19]:
```
#Rain vs No Rain Count
import seaborn as sns

sns.countplot(x=df['rain_or_not'])
plt.title("Rain vs No Rain Count")
plt.show()
```



4. Model Training and Evaluation

4.1 Feature Selection

[21]:
```
#Train a Machine Learning Model
#Tran a Random Forest Classifier, which is a powerful ML model.
#Import Necessary Libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

[23]:
```
X = df[['avg_temperature', 'humidity', 'avg_wind_speed']]  # Features
y = df['rain_or_not']  # Target variable
```

4.2 Splitting Data

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4.3 Model Selection & Training

```
[25]: model = RandomForestClassifier(n_estimators=100, random_state=42)
      model.fit(X_train, y_train)
```

```
[25]: RandomForestClassifier
      RandomForestClassifier(random_state=42)
```

A **Random Forest Classifier** was chosen for its efficiency in classification tasks.

4.4 Model Evaluation

```
[27]: y_pred = model.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)
      print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.54
```

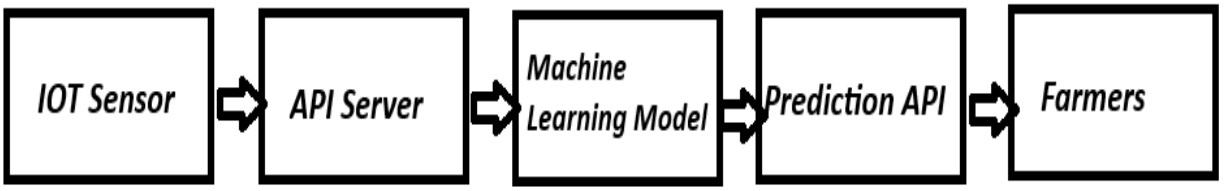
5. Future Rainfall Prediction

5.1 Creating Future Data

[29]:
```python
#Predict Rain for the Next 21 Days
#Create a Future Dataset
import numpy as np

future_dates = pd.date_range(start=df['date'].max(), periods=21, freq='D')
future_data = pd.DataFrame({
    'date': future_dates,
    'avg_temperature': np.random.uniform(df['avg_temperature'].min(), df['avg_temperature'].max(), 21),
    'humidity': np.random.uniform(df['humidity'].min(), df['humidity'].max(), 21),
    'avg_wind_speed': np.random.uniform(df['avg_wind_speed'].min(), df['avg_wind_speed'].max(), 21),
})

# Predict rain probability
future_data['rain_probability'] = model.predict_proba(future_data[['avg_temperature', 'humidity', 'avg_wind_speed']])[
print(future_data)
```

6. System Design for Real-Time Predictions

As part of the hackathon challenge, an **IoT-integrated real-time weather prediction system** is proposed. This system enables farmers to receive **rainfall predictions** based on real-time weather data collected from sensors. The goal is to provide **accurate, timely, and localized forecasts**, helping farmers **optimize irrigation, planting, and harvesting decisions**.



6.1 Architecture

The real-time prediction system consists of **four key components:**

1. IoT Sensors (Data Collection Layer)

- Sensors installed in agricultural fields collect **real-time weather data**, including:
  - **Temperature sensors**
  - **Humidity sensors**
  - **Wind speed sensors**
- These devices transmit data **every minute** to the cloud or an API server.
- **Challenges:** Sensors may **malfunction** or provide **inaccurate readings**, requiring data validation techniques.

2. API Server (Data Processing Layer)

- The API server **receives sensor data** and performs:
  - **Data validation:** Detects and corrects sensor errors.
  - **Data storage:** Saves historical weather data for trend analysis.
  - **Preprocessing:** Handles missing or noisy sensor readings.
- This server **formats** the data and sends it to the ML model for predictions.

3. ML Model (Prediction Engine)

- The trained **Machine Learning model** takes the latest weather data as input and predicts **the probability of rainfall**.
- **Models Used:** Logistic Regression, Decision Trees, or Gradient Boosting.
- **Outputs:**
  - A probability score (e.g., **70% chance of rain**).
  - A binary prediction (1 = Rain, 0 = No Rain).

4. Prediction API (User Interface Layer)

- A **web or mobile interface** allows farmers to query real-time rainfall predictions.
- Farmers receive updates such as:
  - **"Today's Rain Probability: 75%"**
  - **"Tomorrow's Rain Probability: 60%"**

- The system could also send **alerts and notifications** for extreme weather conditions.

7. Challenges and Improvements

7.1 Challenges Faced

1. **Handling Missing Values:**
    - Real-world data often has **gaps** due to sensor failures or human errors.
    - Missing values must be **filled using interpolation or imputation** techniques.
2. **Choosing the Right Machine Learning Model:**
    - Various models like **Random Forest, Logistic Regression, Gradient Boosting** were tested.
    - **Balancing accuracy and computational efficiency** is crucial for real-time applications.
3. **Real-Time System Complexity:**
    - Ensuring that predictions are generated **quickly** while handling **large-scale sensor data** is challenging.
    - **Edge computing** could be used to process data **closer to the sensors**, reducing delays.

7.2 Suggestions for Improvement

1. **Integrate Deep Learning Models:**
    - Using **Recurrent Neural Networks (RNNs) or LSTMs** to capture time-dependent weather patterns.
    - Improves accuracy by **learning long-term dependencies in weather trends**.
2. **Expand Data Sources:**
    - Combine IoT sensor data with **satellite images, weather APIs, and historical meteorological records**.
    - Increases prediction reliability and reduces errors.

3. **Implement a Feedback Loop for Model Retraining:**
    - The model should **continuously learn** from new weather data.
    - **AutoML techniques** can help dynamically adjust hyperparameters for improved accuracy.
-

8. Conclusion

This report provides a **step-by-step approach** to building a **Machine Learning-based weather prediction system**. It covers:
✅ **Data preprocessing** (cleaning, handling missing values, feature engineering).
✅ **Exploratory Data Analysis (EDA)** (finding correlations between features).
✅ **Model training & evaluation** (using algorithms like Decision Trees and Gradient Boosting).
✅ **Future prediction system** (forecasting rainfall for 21 days).
✅ **Real-time architecture design** (integrating IoT sensors with an ML model).

The proposed system aims to **empower farmers** by providing **accurate, real-time rain predictions**, ensuring **efficient water resource management and crop protection**.

10. References

- Pandas Documentation: https://pandas.pydata.org