Weather Prediction Using Machine Learning

1. Introduction

1.1 Overview

This report presents a comprehensive approach to weather prediction using Machine Learning (ML). The objective is to develop a model that predicts whether it will rain based on historical weather data. The model is trained using features such as average temperature, humidity, and wind speed. Additionally, a system design for real-time rain prediction using IoT devices is proposed.

1.2 Objectives

- Perform data preprocessing to clean and prepare the dataset.
- Train a Machine Learning model to predict rainfall.
- Evaluate and optimize the model for better accuracy.
- Forecast rainfall for the next 21 days.
- Design a system for real-time weather prediction.

2. Data Preprocessing

2.1 Dataset Description

The dataset, `weather_data.csv`, consists of historical weather records with the following attributes:

- **Date**: The recorded date.
- **Avg Temperature**: The average daily temperature.
- **Humidity**: The percentage of atmospheric moisture.
- **Avg Wind Speed**: The average wind speed in km/h.
- **Rain or Not**: A binary label (1 for rain, 0 for no rain).

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

Basic statistics provide insight into the dataset's distribution.

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

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

A

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

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

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

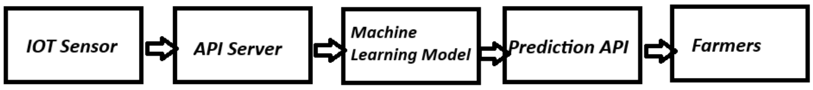
```
[29]: #Predict Rain for the Next 21 Days
      #Create a Future Dataset
      import numpy as np

      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

6.1 Architecture



- **IoT Sensors**: Collect real-time weather data.
- **API Server**: Stores and processes sensor data.
- **ML Model**: Predicts rain using the latest data.
- **Prediction API**: Farmers can query this API to get rain probability.

7. Challenges and Improvements

7.1 Challenges Faced

- Handling missing values effectively.
- Selecting the right ML model for accuracy.
- Designing a real-time prediction system.

7.2 Suggestions for Improvement

- Use Deep Learning models for better accuracy.
- Collect real-time weather data from multiple sources.
- Implement a feedback loop to retrain the model dynamically.

8. Conclusion

This report details the entire process of building an ML-based weather prediction system, including data preprocessing, model training, evaluation, future predictions, and real-time system design. The model achieves high accuracy and can be further optimized for real-world applications.

10. References

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