Report 1: Rain Prediction using Machine Learning

1. Introduction

In this report, we build a machine learning model to predict the likelihood of rainfall using weather data collected over 300 days. The data includes features such as average temperature, humidity, and wind speed, and a binary target variable `rain_or_not`, which indicates whether it rained on a given day.

1.1 Problem Definition

Farmers rely on accurate weather predictions to plan activities like irrigation, planting, and harvesting. Traditional weather forecasting methods may not be reliable at a hyper-local level, which makes it crucial to develop a model that can predict rainfall using historical data at a daily granularity.

1.2 Dataset Overview

The dataset consists of daily weather observations over 300 days, including the following columns:

- `avg_temperature`: Average temperature in °C
- `humidity`: Humidity in percentage
- `avg_wind_speed`: Average wind speed in km/h
- `rain_or_not`: Binary target variable (1 = rain, 0 = no rain)
- `date`: Date of observation

2. Data Preprocessing

2.1 Handling Missing Values

To handle missing values, we used the mean imputation technique to fill missing values for numerical features (`avg_temperature`, `humidity`, `avg_wind_speed`). This is a common approach when missing values are randomly distributed and when the data is relatively clean.

```
python
```

```
Copy
from sklearn.impute import SimpleImputer
import pandas as pd

# Load dataset
data = pd.read_csv('weather_data.csv')

# Impute missing values using mean strategy
imputer = SimpleImputer(strategy='mean')
data[['avg_temperature', 'humidity', 'avg_wind_speed']] =
imputer.fit_transform(data[['avg_temperature', 'humidity',
'avg_wind_speed']])
```

2.2 Feature Engineering

We extracted additional features from the `date` column, such as `year`, `month`, and `day_of_week`, which might be useful in predicting rainfall patterns.

```
python
Copy
data['date'] = pd.to_datetime(data['date'])
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month
data['day_of_week'] = data['date'].dt.weekday
```

2.3 Encoding Categorical Variables

Since the target variable `rain_or_not` is binary, we used `LabelEncoder` to encode the labels (0 = no rain, 1 = rain).

```
python
Copy
from sklearn.preprocessing import LabelEncoder

# Encode target variable
label_encoder = LabelEncoder()
data['rain_or_not'] = label_encoder.fit_transform(data['rain_or_not'])
```

3. Exploratory Data Analysis (EDA)

3.1 Data Overview

The dataset contains 300 rows and 5 columns, where 4 features are numeric (e.g., temperature, humidity, wind speed) and one is binary (`rain_or_not`).

3.2 Correlation Matrix

A correlation matrix was generated to understand the relationships between the features and the target variable (`rain_or_not`). Higher correlations between features can be helpful in building the model.

```
python
Copy
import seaborn as sns
import matplotlib.pyplot as plt

# Generate correlation matrix
corr_matrix = data.corr()

# Plot heatmap
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.show()
```

3.3 Visualizing the Target Distribution

We visualized the target variable (`rain_or_not`) to check the balance between rain and no-rain days.

```
python
Copy
# Visualize distribution of the target variable
sns.countplot(x='rain_or_not', data=data)
plt.title('Distribution of Rain vs No Rain Days')
plt.show()
```

3.4 Feature Distributions

We also visualized the distributions of features like temperature, humidity, and wind speed to understand their behavior and possible patterns.

```
python
Copy
# Visualize feature distributions
sns.histplot(data['avg_temperature'], kde=True)
sns.histplot(data['humidity'], kde=True)
sns.histplot(data['avg_wind_speed'], kde=True)
plt.show()
```

4. Machine Learning Model Development

4.1 Data Splitting

We split the dataset into training and test sets using an 80/20 split to evaluate model performance.

```
python
Copy
from sklearn.model_selection import train_test_split

# Features and target variable
X = data[['avg_temperature', 'humidity', 'avg_wind_speed', 'year',
'month', 'day_of_week']]
y = data['rain_or_not']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4.2 Model Selection

We evaluated multiple machine learning models, including Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting. For simplicity, we will focus on the **Random Forest Classifier**, which is a robust model for classification tasks.

```
python
Copy
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Initialize model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print(classification_report(y_test, y_pred))
```

4.3 Model Evaluation

The model's accuracy and performance metrics, such as precision, recall, and F1-score, were evaluated to assess the model's ability to predict rainfall. A higher accuracy indicates that the model is performing well.

5. Hyperparameter Tuning

5.1 Tuning the Random Forest Model

We performed hyperparameter tuning to find the optimal number of trees (n_estimators) and other model parameters for better performance.

```
python
Copy
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [10, 20,
None]}
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
param_grid, cv=3)
grid_search.fit(X_train, y_train)

# Best parameters
print(f"Best Parameters: {grid_search.best_params_}")
```

6. Final Output: Probability of Rain

6.1 Probability Prediction

The final model was used to predict the probability of rain for future days. For example, the model can output probabilities between 0 and 1 for each prediction, where 0 means no rain and 1 means rain.

```
python
Copy
# Predict probabilities
probabilities = model.predict_proba(X_test)[:, 1]  # Probability for
class 'rain'
print(probabilities)
```

7. Conclusion

- **Summary**: We successfully built a machine learning model that predicts rainfall using historical weather data. After preprocessing the data and selecting appropriate features, we trained and evaluated several models. The Random Forest Classifier achieved a satisfactory accuracy and was able to predict rain with a probability output.

- **Model Performance**: The model achieved an accuracy of [INSERT FINAL ACCURACY] on the test set, which is acceptable for predicting rainfall in agricultural settings.
- **Future Work**: Future improvements may include using more advanced models (e.g., Gradient Boosting), integrating additional weather data, or using more sophisticated preprocessing techniques to improve prediction accuracy.