Notebook

June 2, 2025

[5]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load and prepare the data
def load_weather_data(data_string):


    # Create DataFrame
    df = pd.read_excel(data_string)

    # Convert columns to appropriate data types
    numeric_cols = [
        'last_updated_epoch', 'temperature_celsius', 'temperature_fahrenheit',
        'wind_mph', 'wind_kph', 'wind_degree', 'pressure_mb', 'pressure_in',
        'precip_mm', 'precip_in', 'humidity', 'cloud', 'feels_like_celsius',
        'feels_like_fahrenheit', 'visibility_km', 'visibility_miles',␣
↪'uv_index',
        'gust_mph', 'gust_kph'
    ]

    for col in numeric_cols:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')

    # Convert timestamp to datetime
    if 'last_updated_epoch' in df.columns:
        df['datetime'] = pd.to_datetime(df['last_updated_epoch'], unit='s')

    return df
```

```python
# Sample data string - replace this with your actual data in production
data_string = r"C:\Users\arjun\Downloads\Weather data.xlsx\Weather data.xlsx"

# Load the data
weather_data = load_weather_data(data_string)

# Step 2: Define weather conditions based on available features
def determine_weather_condition(row):
    """
    Determine weather condition based on available meteorological data

    Rules:
    - Rainy: any precipitation or very high humidity with high cloud cover
    - Cloudy: high cloud cover but no precipitation
    - Sunny: low cloud cover
    """
    if row['precip_mm'] > 0:
        return 0  # Rainy
    elif row['cloud'] > 70:
        return 1  # Cloudy
    else:
        return 2  # Sunny

# Add weather condition to dataset
weather_data['weather_condition'] = weather_data.
 ↪apply(determine_weather_condition, axis=1)

# Add weather condition labels
weather_labels = {0: 'Rainy', 1: 'Cloudy', 2: 'Sunny'}
weather_data['weather_label'] = weather_data['weather_condition'].
 ↪map(weather_labels)

# Create target variable (next hour's weather)
weather_data['next_hour_weather'] = weather_data['weather_condition'].shift(-1)
weather_data['next_hour_weather_label'] = weather_data['weather_label'].
 ↪shift(-1)

# Drop the last row which has NaN in the target variable
weather_data = weather_data.dropna(subset=['next_hour_weather'])

# Display the first few rows of the dataset
print("Sample of the processed weather dataset:")
print(weather_data[['datetime', 'temperature_celsius', 'humidity', 'cloud',
                   'precip_mm', 'weather_label', 'next_hour_weather_label']].
 ↪head())

# Step 3: Feature Engineering
```

```
# Extract time features
weather_data['hour'] = weather_data['datetime'].dt.hour
weather_data['day'] = weather_data['datetime'].dt.day
weather_data['month'] = weather_data['datetime'].dt.month
weather_data['day_of_week'] = weather_data['datetime'].dt.dayofweek

# Calculate differences from previous hour
weather_data['temp_diff'] = weather_data['temperature_celsius'].diff().fillna(0)
weather_data['humidity_diff'] = weather_data['humidity'].diff().fillna(0)
weather_data['pressure_diff'] = weather_data['pressure_mb'].diff().fillna(0)
weather_data['cloud_diff'] = weather_data['cloud'].diff().fillna(0)

# Calculate rolling averages (3-hour)
weather_data['temp_3h_avg'] = weather_data['temperature_celsius'].
 ↪rolling(window=3).mean().fillna(weather_data['temperature_celsius'])
weather_data['humidity_3h_avg'] = weather_data['humidity'].rolling(window=3).
 ↪mean().fillna(weather_data['humidity'])
weather_data['pressure_3h_avg'] = weather_data['pressure_mb'].rolling(window=3).
 ↪mean().fillna(weather_data['pressure_mb'])
weather_data['cloud_3h_avg'] = weather_data['cloud'].rolling(window=3).mean().
 ↪fillna(weather_data['cloud'])

# Step 4: Exploratory Data Analysis
# Check distribution of target variable
print("\nDistribution of weather conditions:")
print(weather_data['next_hour_weather_label'].value_counts())

# Create visualizations
plt.figure(figsize=(10, 6))
sns.boxplot(x='weather_label', y='temperature_celsius', data=weather_data)
plt.title('Temperature Distribution by Weather Condition')
plt.savefig('temp_by_weather.png')
plt.close()

plt.figure(figsize=(10, 6))
sns.boxplot(x='weather_label', y='humidity', data=weather_data)
plt.title('Humidity Distribution by Weather Condition')
plt.savefig('humidity_by_weather.png')
plt.close()

plt.figure(figsize=(10, 6))
sns.boxplot(x='weather_label', y='cloud', data=weather_data)
plt.title('Cloud Cover Distribution by Weather Condition')
plt.savefig('cloud_by_weather.png')
plt.close()

# Create correlation matrix for numeric features
```

```
plt.figure(figsize=(16, 12))
numeric_columns = ['temperature_celsius', 'humidity', 'pressure_mb', 'wind_mph',
                   'cloud', 'precip_mm', 'uv_index', 'gust_mph',
                   'weather_condition', 'next_hour_weather']
correlation_data = weather_data[numeric_columns].corr()
sns.heatmap(correlation_data, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Weather Features')
plt.tight_layout()
plt.savefig('correlation_matrix.png')
plt.close()

# Step 5: Prepare data for modeling
feature_columns = [
    'temperature_celsius', 'humidity', 'pressure_mb', 'wind_mph', 'cloud',
    'precip_mm', 'visibility_km', 'uv_index', 'gust_mph', 'weather_condition',
    'hour', 'day_of_week', 'month', 'temp_diff', 'humidity_diff',
 ↪'pressure_diff',
    'cloud_diff', 'temp_3h_avg', 'humidity_3h_avg', 'pressure_3h_avg',
 ↪'cloud_3h_avg'
]

X = weather_data[feature_columns]
y = weather_data['next_hour_weather']

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

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 6: Train the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# Step 7: Evaluate the model
y_pred = rf_model.predict(X_test_scaled)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred, target_names=['Rainy',
 ↪'Cloudy', 'Sunny'])

print("\nModel Evaluation:")
```

```
print(f"Accuracy: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Rainy', 'Cloudy', 'Sunny'],
            yticklabels=['Rainy', 'Cloudy', 'Sunny'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.savefig('confusion_matrix.png')
plt.close()

# Feature importance
feature_importance = pd.DataFrame({
    'Feature': feature_columns,
    'Importance': rf_model.feature_importances_
}).sort_values('Importance', ascending=False)

print("\nFeature Importance:")
print(feature_importance.head(10))  # Show top 10 features

plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance.head(15))
plt.title('Top 15 Feature Importance')
plt.tight_layout()
plt.savefig('feature_importance.png')
plt.close()

# Step 8: Function to predict weather for new data
def predict_next_hour_weather(model, scaler, feature_values):
    """
    Predict the next hour's weather based on current conditions

    Parameters:
    - model: trained classifier model
    - scaler: fitted StandardScaler
    - feature_values: dictionary of feature values matching the model's expected features

    Returns:
    - predicted weather condition as string ('Rainy', 'Cloudy', or 'Sunny')
```

```
    - probabilities for each class
    """
    # Create a dataframe with the input features
    input_data = pd.DataFrame([feature_values])

    # Ensure all required features are present
    for col in feature_columns:
        if col not in input_data.columns:
            input_data[col] = 0  # Set missing features to 0

    # Make sure columns are in the correct order
    input_data = input_data[feature_columns]

    # Scale the input data
    input_data_scaled = scaler.transform(input_data)

    # Make prediction
    prediction = model.predict(input_data_scaled)[0]
    probabilities = model.predict_proba(input_data_scaled)[0]

    # Convert numerical prediction to label
    weather_labels = {0: 'Rainy', 1: 'Cloudy', 2: 'Sunny'}
    predicted_weather = weather_labels[prediction]

    return predicted_weather, probabilities

# Example: Current weather data
example_weather = {
    'temperature_celsius': 28.5,
    'humidity': 65,
    'pressure_mb': 1008,
    'wind_mph': 12.0,
    'cloud': 30,
    'precip_mm': 0,
    'visibility_km': 10,
    'uv_index': 7,
    'gust_mph': 14.0,
    'weather_condition': 2,  # Current: Sunny
    'hour': 14,
    'day_of_week': 3,  # Thursday
    'month': 8,  # August
    'temp_diff': 0.5,
    'humidity_diff': -2.0,
    'pressure_diff': -1.0,
    'cloud_diff': 5.0,
    'temp_3h_avg': 28.0,
    'humidity_3h_avg': 66.0,
```

```python
    'pressure_3h_avg': 1008.5,
    'cloud_3h_avg': 28.0
}

predicted_weather, probabilities = predict_next_hour_weather(rf_model, scaler,␣
 ↪example_weather)

print(f"\nPredicted weather for next hour: {predicted_weather}")
print(f"Prediction probabilities: Rainy: {probabilities[0]:.2f}, Cloudy:␣
 ↪{probabilities[1]:.2f}, Sunny: {probabilities[2]:.2f}")

# Step 9: Save the model
import joblib

# Save the model and scaler
joblib.dump(rf_model, 'hourly_weather_model.pkl')
joblib.dump(scaler, 'hourly_weather_scaler.pkl')

print("\nModel and scaler saved successfully!")
```

```
Sample of the processed weather dataset:
             datetime  temperature_celsius  humidity  cloud  precip_mm  \
0 2023-08-25 21:46:41                 27.5        67     26        0.0
1 2023-08-25 21:46:42                 27.5        70     19        0.0
2 2023-08-25 21:46:43                 26.3        70     51        0.0
3 2023-08-25 21:46:44                 25.6        76     65        0.0
4 2023-08-25 21:46:45                 27.2        74     82        0.0

  weather_label next_hour_weather_label
0         Sunny                   Sunny
1         Sunny                   Sunny
2         Sunny                   Sunny
3         Sunny                  Cloudy
4        Cloudy                   Sunny

Distribution of weather conditions:
next_hour_weather_label
Sunny     14479
Rainy      7354
Cloudy     2236
Name: count, dtype: int64

Model Evaluation:
Accuracy: 0.6961

Confusion Matrix:
[[ 765   28  627]
```

```
 [ 168   54  221]
 [ 398   21 2532]]

Classification Report:
              precision    recall  f1-score   support

       Rainy       0.57      0.54      0.56      1420
      Cloudy       0.52      0.12      0.20       443
       Sunny       0.75      0.86      0.80      2951

    accuracy                           0.70      4814
   macro avg       0.62      0.51      0.52      4814
weighted avg       0.68      0.70      0.67      4814


Feature Importance:
                Feature  Importance
20         cloud_3h_avg    0.117155
4                 cloud    0.084957
18      humidity_3h_avg    0.084219
17          temp_3h_avg    0.064452
16           cloud_diff    0.064283
8              gust_mph    0.060353
13            temp_diff    0.058491
0   temperature_celsius    0.057158
1              humidity    0.053519
14        humidity_diff    0.052484

Predicted weather for next hour: Sunny
Prediction probabilities: Rainy: 0.10, Cloudy: 0.02, Sunny: 0.88

Model and scaler saved successfully!
```

[7]:

```python
def interactive_hourly_prediction():
    """
    Interactive function to input current weather data and predict next hour's␣
 ↪weather
    """
    print("\n===== Hourly Weather Prediction Tool =====")
    print("Enter current weather data to predict weather for the next hour")

    # Get user input for key weather metrics
    temp = float(input("Current temperature (°C): "))
    humidity = float(input("Current humidity (%): "))
    pressure = float(input("Current pressure (mb): "))
    wind = float(input("Current wind speed (mph): "))
    cloud = float(input("Current cloud cover (0-100%): "))
```

```python
    precip = float(input("Current precipitation (mm): "))

    # Set up the prediction input
    current_hour = int(input("Current hour of day (0-23): "))
    current_day = int(input("Current day of week (0=Monday, 6=Sunday): "))
    current_month = int(input("Current month (1-12): "))

    # Determine current weather condition
    if precip > 0:
        current_condition = 0  # Rainy
    elif cloud > 70:
        current_condition = 1  # Cloudy
    else:
        current_condition = 2  # Sunny

    # Create prediction input
    prediction_input = {
        'temperature_celsius': temp,
        'humidity': humidity,
        'pressure_mb': pressure,
        'wind_mph': wind,
        'cloud': cloud,
        'precip_mm': precip,
        'visibility_km': 10,  # Default value
        'uv_index': 5,  # Default value
        'gust_mph': wind * 1.2,  # Estimate gust as 20% higher than wind speed
        'weather_condition': current_condition,
        'hour': current_hour,
        'day_of_week': current_day,
        'month': current_month,
        'temp_diff': 0,  # Assuming we don't have previous data
        'humidity_diff': 0,
        'pressure_diff': 0,
        'cloud_diff': 0,
        'temp_3h_avg': temp,  # Using current values as approximation
        'humidity_3h_avg': humidity,
        'pressure_3h_avg': pressure,
        'cloud_3h_avg': cloud
    }

    # Load the model and make prediction
    loaded_model = joblib.load('hourly_weather_model.pkl')
    loaded_scaler = joblib.load('hourly_weather_scaler.pkl')

    prediction, probs = predict_next_hour_weather(loaded_model, loaded_scaler,␣
↪prediction_input)
```

```python
    # Display results
    print("\n===== Prediction Results =====")
    print(f"Weather for the next hour is predicted to be: {prediction}")
    print(f"Prediction confidence:")
    print(f"  Rainy:  {probs[0]:.2f} ({probs[0]*100:.1f}%)")
    print(f"  Cloudy: {probs[1]:.2f} ({probs[1]*100:.1f}%)")
    print(f"  Sunny:  {probs[2]:.2f} ({probs[2]*100:.1f}%)")

    if max(probs) < 0.5:
        print("\nNote: This prediction has low confidence.")

# Uncomment to use the interactive prediction tool
interactive_hourly_prediction()
```

```
===== Hourly Weather Prediction Tool =====
Enter current weather data to predict weather for the next hour

Current temperature (°C):  29
Current humidity (%):  94
Current pressure (mb):  29.74
Current wind speed (mph):  12
Current cloud cover (0-100%):  71
Current precipitation (mm):  0.00
Current hour of day (0-23):  9
Current day of week (0=Monday, 6=Sunday):  6
Current month (1-12):  5


===== Prediction Results =====
Weather for the next hour is predicted to be: Sunny
Prediction confidence:
  Rainy:  0.34 (34.0%)
  Cloudy: 0.24 (24.0%)
  Sunny:  0.42 (42.0%)

Note: This prediction has low confidence.
```

[ ]:

```python

```

This notebook was converted with convert.ploomber.io