alert-generation-system

September 5, 2024

Machine Learning Model For Alert Generation

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Generate synthetic dataset (same as before)
n_samples = 1000
temperature_range = (0, 30)
precipitation_range = (0, 200)
seismic_activity_range = (0, 10)
water_level_range = (0, 100)

np.random.seed(42)

temperature = np.random.uniform(temperature_range[0], temperature_range[1],␣
 ↪n_samples)
precipitation = np.random.uniform(precipitation_range[0],␣
 ↪precipitation_range[1], n_samples)
seismic_activity = np.random.uniform(seismic_activity_range[0],␣
 ↪seismic_activity_range[1], n_samples)
water_level = np.random.uniform(water_level_range[0], water_level_range[1],␣
 ↪n_samples)

# Initialize the target variable and alert levels
glof_occurred = []
alert_signal = []

# Define thresholds for alert levels
for i in range(n_samples):
    # GLOF occurrence logic
    if (temperature[i] > 15 and precipitation[i] > 150 and
        seismic_activity[i] > 5 and water_level[i] > 70):
        glof_occurred.append(1)  # GLOF occurred

        # Red alert (Evacuate immediately)
```

```
        if (temperature[i] > 25 and precipitation[i] > 180 and
            seismic_activity[i] > 8 and water_level[i] > 90):
            alert_signal.append('Red')
        # Orange alert (High Alert)
        elif (temperature[i] > 20 and precipitation[i] > 160 and
              seismic_activity[i] > 6.5 and water_level[i] > 80):
            alert_signal.append('Orange')
        # Yellow alert (Caution)
        else:
            alert_signal.append('Yellow')
    else:
        glof_occurred.append(0)  # No GLOF
        alert_signal.append('Green')  # Safe condition

# Create DataFrame
data = pd.DataFrame({
    'temperature': temperature,
    'precipitation': precipitation,
    'seismic_activity': seismic_activity,
    'water_level': water_level,
    'alert_signal': alert_signal
})

# Features (X) and target (y)
X = data[['temperature', 'precipitation', 'seismic_activity', 'water_level']]
y = data['alert_signal']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train a Random Forest Classifier
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.995
Classification Report:
              precision    recall  f1-score   support
```

```
       Green       0.99      1.00      1.00       199
      Yellow       0.00      0.00      0.00         1

    accuracy                           0.99       200
   macro avg       0.50      0.50      0.50       200
weighted avg       0.99      0.99      0.99       200

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

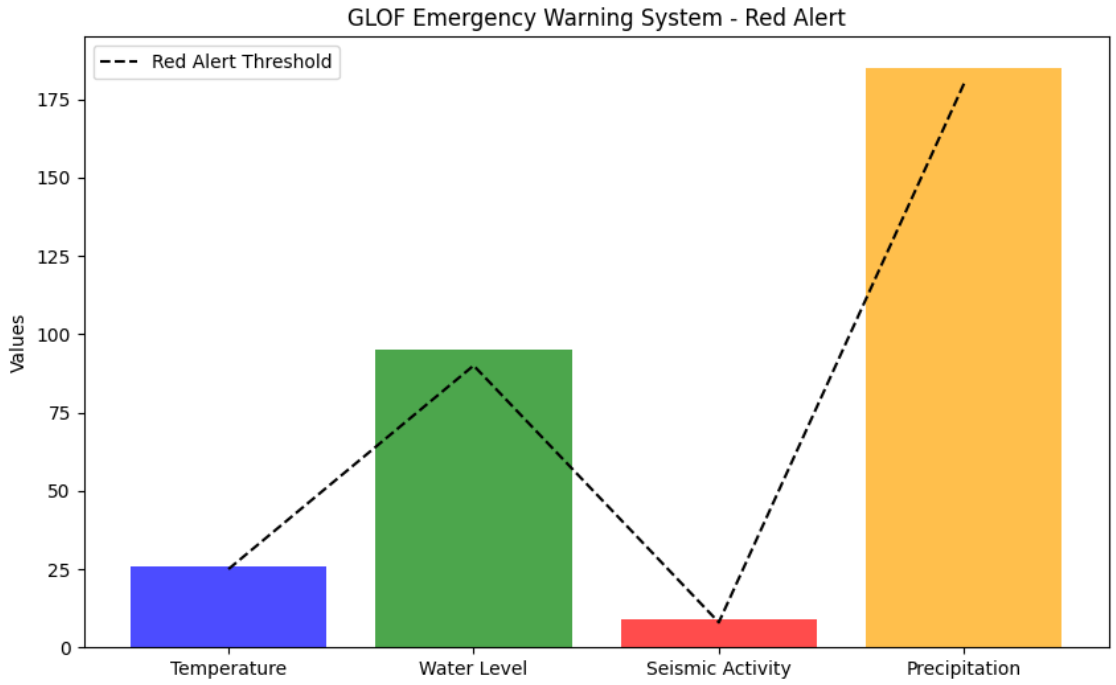
[ ]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Define threshold values for the alert system
def get_alert_level(temperature, water_level, seismic_activity, precipitation):
    """
    Determines the alert level based on the sensor inputs.

    Returns:
    - alert_level (str): Red, Orange, Yellow, Green
    - precautions (str): Appropriate message based on the alert level
    """
    # Red Alert: Critical (Evacuate Immediately)
    if temperature > 25 and precipitation > 180 and seismic_activity > 8 and␣
↪water_level > 90:
        return 'Red', (
            "Critical Alert: Evacuate immediately. Seismic activity and water␣
↪levels are dangerously high. "
            "Prepare for potential GLOF. Follow evacuation routes and listen to␣
↪local authorities."
        )

    # Orange Alert: High Risk (High Alert)
```

```
    elif temperature > 20 and precipitation > 160 and seismic_activity > 6.5␣
↪and water_level > 80:
        return 'Orange', (
            "High Alert: Prepare for possible evacuation. Water levels and␣
↪seismic activity are elevated. "
            "Stay informed and be ready to evacuate if conditions worsen."
        )

    # Yellow Alert: Moderate Risk (Caution)
    elif temperature > 15 and precipitation > 150 and seismic_activity > 5 and␣
↪water_level > 70:
        return 'Yellow', (
            "Caution: Monitor conditions closely. Elevated water levels and␣
↪seismic activity. "
            "Stay alert and avoid dangerous areas around the lake or river."
        )

    # Green Alert: Safe
    else:
        return 'Green', "Safe: Conditions are normal. No immediate risk of GLOF.
↪ Stay informed for any updates."

# Function to generate the emergency warning system
def emergency_warning_system(temp, water_lvl, seismic_act, precip):
    """
    Takes in the four parameters (temperature, water level, seismic activity,␣
↪and precipitation)
    and returns the alert level, precaution message, and visual representation␣
↪of the data.
    """
    # Get the alert level and precautions
    alert, precautions = get_alert_level(temp, water_lvl, seismic_act, precip)

    # Print the alert level and precautions
    print(f"Alert Level: {alert}")
    print(f"Precaution: {precautions}\n")

    # Create a DataFrame to hold the data
    data = {
        'Parameter': ['Temperature', 'Water Level', 'Seismic Activity',␣
↪'Precipitation'],
        'Value': [temp, water_lvl, seismic_act, precip],
        'Threshold': [25, 90, 8, 180]  # Red alert threshold
    }
    df = pd.DataFrame(data)
```

```
    # Plot the data
    plt.figure(figsize=(10, 6))
    plt.bar(df['Parameter'], df['Value'], color=['blue', 'green', 'red',␣
 ↪'orange'], alpha=0.7)
    plt.plot(df['Parameter'], df['Threshold'], color='black', linestyle='--',␣
 ↪label='Red Alert Threshold')
    plt.title(f"GLOF Emergency Warning System - {alert} Alert")
    plt.ylabel('Values')
    plt.legend()
    plt.ylim(0, max(df['Value'].max(), df['Threshold'].max()) + 10)  # Adjust␣
 ↪y-axis for better visualization
    plt.show()

# Example input data (you can modify these to test different scenarios)
temperature = 26  # in Celsius
water_level = 95  # in meters
seismic_activity = 9  # Richter scale
precipitation = 185  # in mm

# Run the emergency warning system
emergency_warning_system(temperature, water_level, seismic_activity,␣
 ↪precipitation)
```

```
Alert Level: Red
Precaution: Critical Alert: Evacuate immediately. Seismic activity and water
levels are dangerously high. Prepare for potential GLOF. Follow evacuation
routes and listen to local authorities.
```

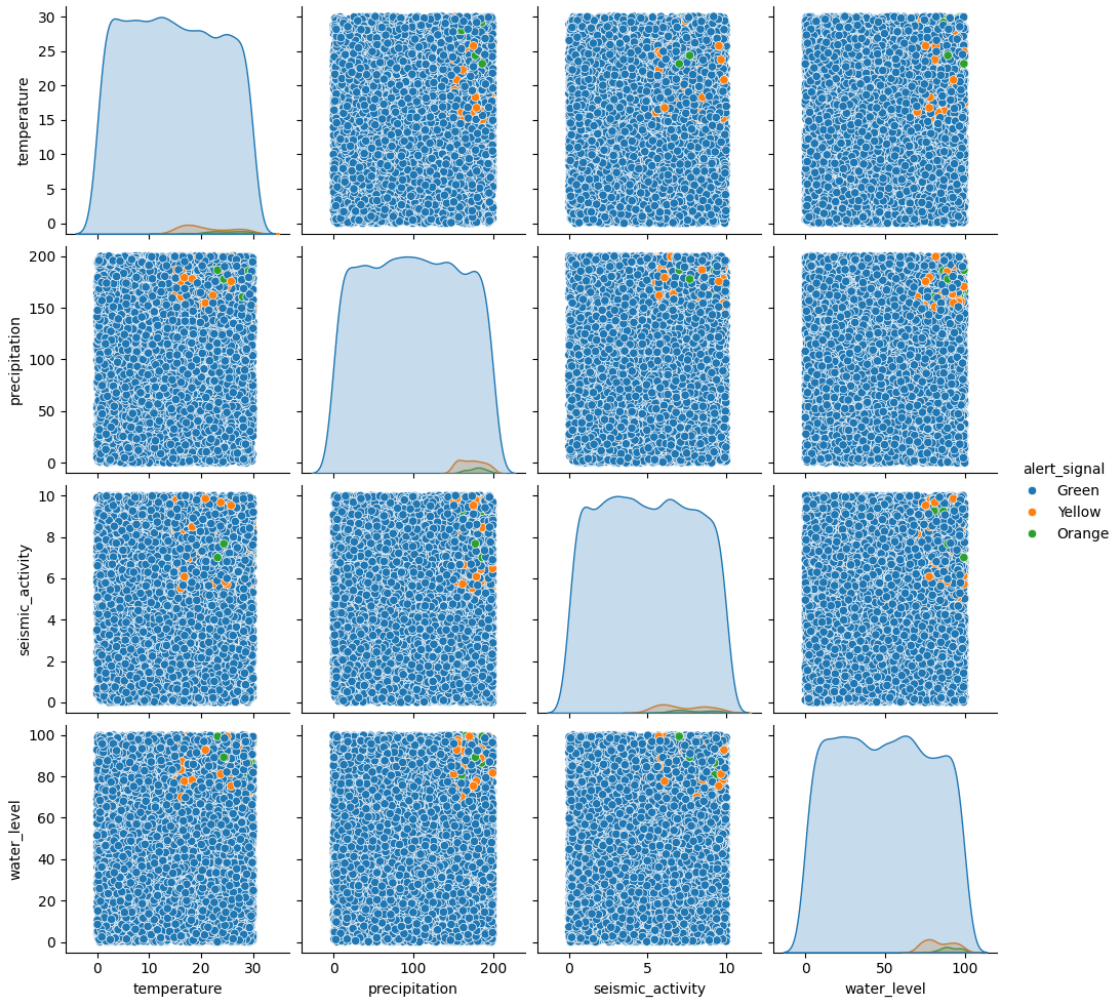


[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Number of samples
n_samples = 10000

# Feature ranges
temperature_range = (0, 30)  # in degrees Celsius
precipitation_range = (0, 200)  # in mm
seismic_activity_range = (0, 10)  # Richter scale
water_level_range = (0, 100)  # in meters

# Set random seed for reproducibility
np.random.seed(42)

# Generate random data for each feature
temperature = np.random.uniform(temperature_range[0], temperature_range[1],␣
 ↪n_samples)
precipitation = np.random.uniform(precipitation_range[0],␣
 ↪precipitation_range[1], n_samples)
```

```
seismic_activity = np.random.uniform(seismic_activity_range[0],␣
 ↪seismic_activity_range[1], n_samples)
water_level = np.random.uniform(water_level_range[0], water_level_range[1],␣
 ↪n_samples)

# Initialize the target variable and alert levels
alert_signal = []

# Define thresholds for alert levels
for i in range(n_samples):
    if (temperature[i] > 25 and precipitation[i] > 180 and seismic_activity[i]␣
 ↪> 8 and water_level[i] > 90):
        alert_signal.append('Red')  # Evacuate immediately
    elif (temperature[i] > 20 and precipitation[i] > 160 and␣
 ↪seismic_activity[i] > 6.5 and water_level[i] > 80):
        alert_signal.append('Orange')  # High Alert
    elif (temperature[i] > 15 and precipitation[i] > 150 and␣
 ↪seismic_activity[i] > 5 and water_level[i] > 70):
        alert_signal.append('Yellow')  # Caution
    else:
        alert_signal.append('Green')  # Safe

# Create a DataFrame
data = pd.DataFrame({
    'temperature': temperature,
    'precipitation': precipitation,
    'seismic_activity': seismic_activity,
    'water_level': water_level,
    'alert_signal': alert_signal
})

# Display the first few rows
print(data.head())

# Plot the data to visualize the feature distributions
sns.pairplot(data, hue='alert_signal')
plt.show()

# Features (X) and target (y)
X = data[['temperature', 'precipitation', 'seismic_activity', 'water_level']]
y = data['alert_signal']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

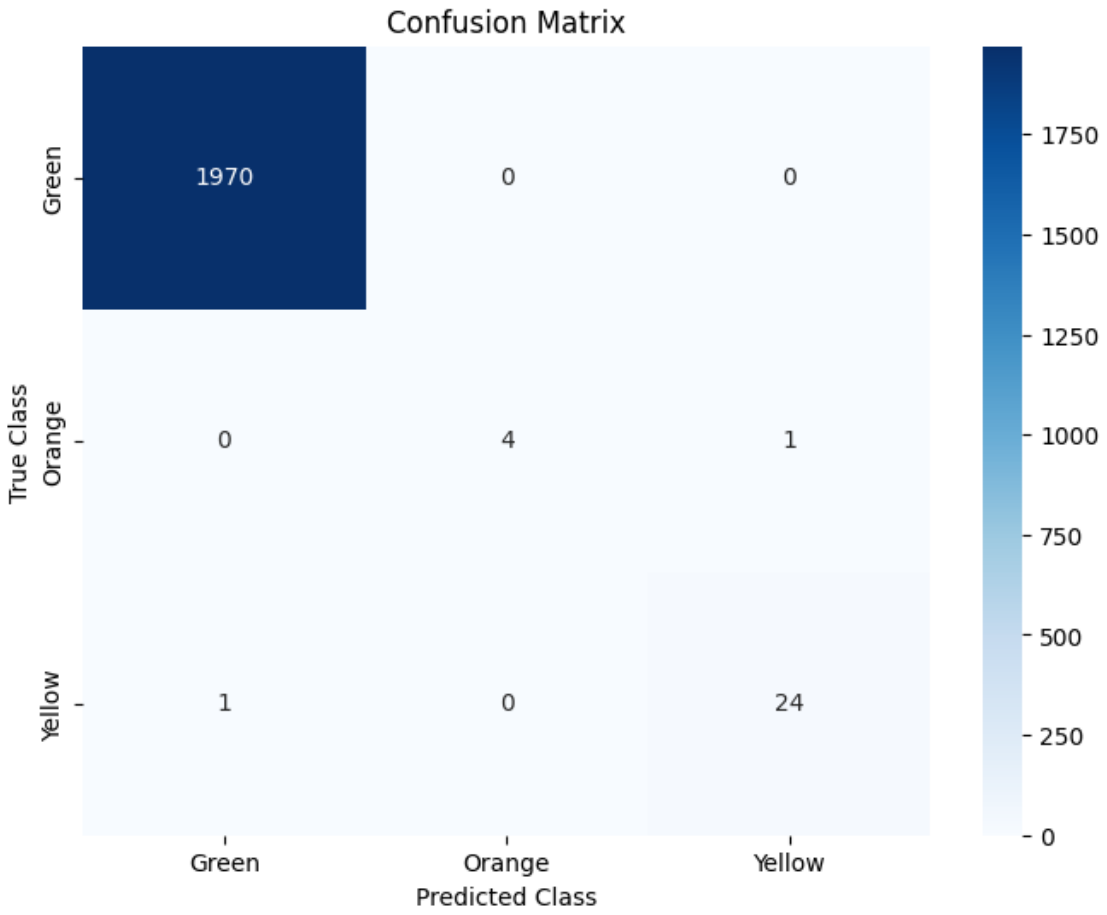
```
# Initialize and train the Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluation: Classification Report and Confusion Matrix
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Plot Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=clf.
 ↪classes_, yticklabels=clf.classes_)
plt.title("Confusion Matrix")
plt.ylabel('True Class')
plt.xlabel('Predicted Class')
plt.show()

# Plot feature importance
importances = clf.feature_importances_
indices = np.argsort(importances)[::-1]
features = X.columns

plt.figure(figsize=(8, 6))
plt.title("Feature Importance")
plt.bar(range(X.shape[1]), importances[indices], color='r', align='center')
plt.xticks(range(X.shape[1]), [features[i] for i in indices], rotation=45)
plt.show()
```

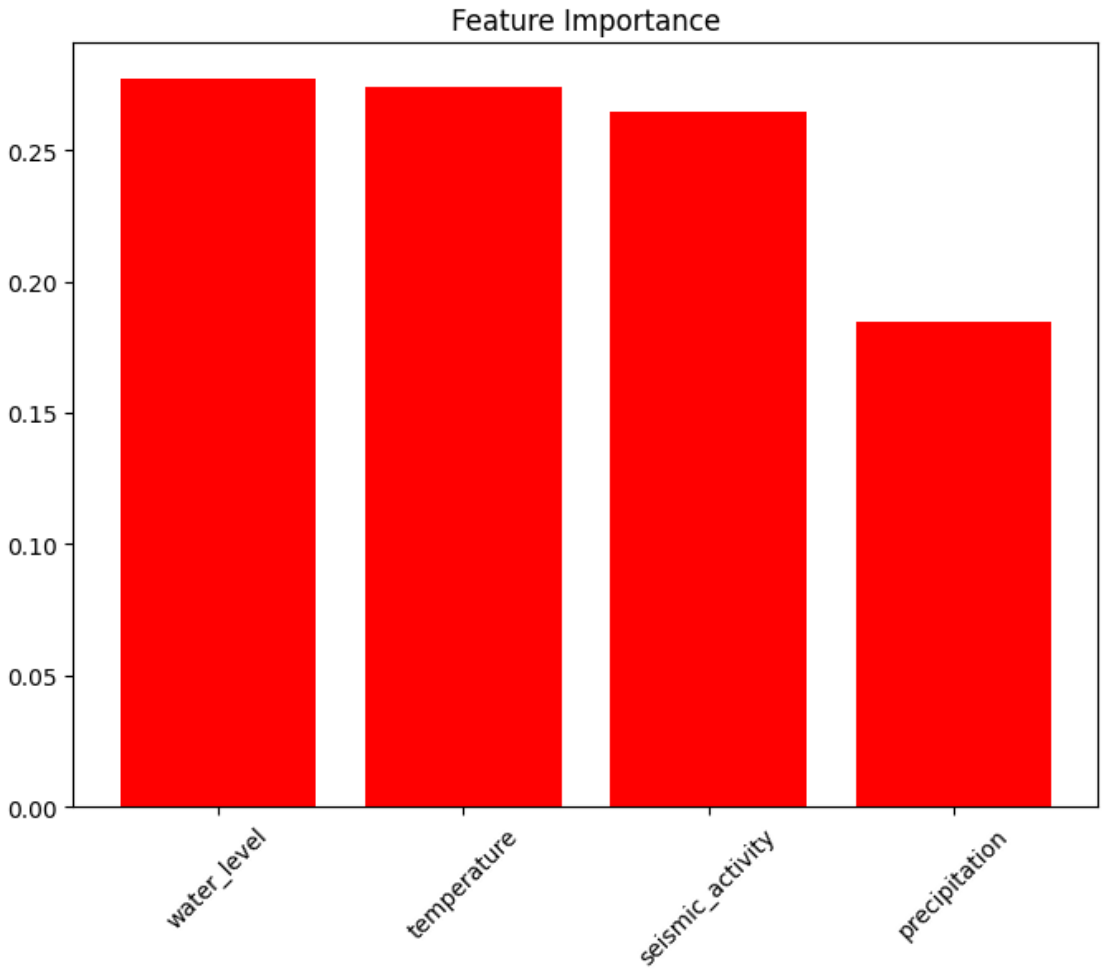
	temperature	precipitation	seismic_activity	water_level	alert_signal
0	11.236204	74.728164	7.299983	63.814457	Green
1	28.521429	66.582419	1.845120	45.929245	Green
2	21.959818	35.230783	3.466397	96.449852	Green
3	17.959755	121.453334	6.632806	21.897845	Green
4	4.680559	95.324832	4.820893	58.785642	Green



Classification Report:

	precision	recall	f1-score	support
Green	1.00	1.00	1.00	1970
Orange	1.00	0.80	0.89	5
Yellow	0.96	0.96	0.96	25
accuracy			1.00	2000
macro avg	0.99	0.92	0.95	2000
weighted avg	1.00	1.00	1.00	2000


Confusion Matrix
Green
Orange
Yellow
True Class
1970
0
0
0
4
1
1
0
24
Green
Orange
Yellow
Predicted Class
1750
1500
1250
1000
750
500
250
0


Feature Importance
0.25
0.20
0.15
0.10
0.05
0.00
water_level
temperature
seismic_activity
precipitation