hail storm prediction model using meteosat-9 MSG HRIT LRIT Data

Hail Storm Prediction Model Using Meteosat-9 MSG HRIT/LRIT Data

Overview

This model aims to predict hail storms using data from Meteosat-9's Spinning Enhanced Visible and Infrared Imager (SEVIRI) instrument, which provides High Rate Information Transmission (HRIT) and Low Rate Information Transmission (LRIT) data.

Key Data Parameters

Essential MSG Channels for Hail Detection:

1. **VIS0.6** (Visible) - Cloud structure analysis
2. **IR3.9** (Near IR) - Night-time fog detection, fire monitoring
3. **WV6.2** (Water Vapor) - Upper-level humidity
4. **WV7.3** (Water Vapor) - Mid-level humidity
5. **IR8.7** (IR Window) - Cloud phase discrimination
6. **IR10.8** (IR Window) - Cloud top temperature
7. **IR12.0** (IR Window) - Cloud particle size estimation
8. **HRV** (High Resolution Visible) - Detailed cloud structure

Model Architecture

1. Data Preprocessing

* **Radiometric calibration**: Convert digital counts to physical units
* **Geometric correction**: Map data to geographic coordinates
* **Temporal alignment**: Synchronize multi-channel data
* **Cloud masking**: Identify convective cloud areas

2. Feature Extraction

* **Cloud top temperature** (from IR10.8)
* **Cloud top height** estimation
* **Cloud phase** discrimination (using IR8.7 and IR10.8 difference)
* **Vertical development** (using HRV and IR difference)
* **Overshooting tops** detection
* **Brightness temperature differences** (BTD):
  + BTD(IR3.9-IR10.8) for hail signature
  + BTD(IR10.8-IR12.0) for particle size estimation

3. Hail Prediction Algorithms

a) Convective Potential Index (CPI)

Copy

Download

CPI = (IR10.8 - WV6.2) + (WV6.2 - WV7.3)

* High CPI values indicate strong updrafts favorable for hail

b) Hail Detection Algorithm (HDA)

Copy

Download

HDA = a\*BTD(IR3.9-IR10.8) + b\*BTD(IR10.8-IR12.0) + c\*CloudTopHeight + d\*CloudTopTemperature

(where a,b,c,d are empirically derived coefficients)

c) Machine Learning Approach

* Random Forest or CNN trained on historical hail events
* Input features:
  + Multi-spectral channel differences
  + Temporal trends (cloud growth rates)
  + Spatial texture features

4. Output Products

* Hail probability maps (0-100%)
* Hail size estimation (small, medium, large)
* Storm tracking and nowcasting (30-90 minute lead time)

Validation Methods

* Comparison with ground radar data
* Verification with hail pad networks
* Match-up with ground reports
* Skill scores (POD, FAR, CSI)

Implementation Considerations

* Temporal resolution: 5-15 minute updates
* Spatial resolution: 3km at nadir (HRIT), 1km for HRV
* Processing chain optimization for real-time operation
* Integration with NWP model outputs

Limitations

* Difficulty with small hail detection
* Challenges in nocturnal hail detection
* Overestimation in dry atmospheric conditions
* Dependence on accurate cloud top property retrievals

This framework provides a comprehensive approach to hail storm prediction using Meteosat-9 data, combining physical retrievals with statistical and machine learning techniques for optimal performance.

give python codes with Graphical User Interface and mapping and plotting at real time and alarm generate in audio and on GUI with percentage probability and size of hail at Islamabad and its 100 kilometre surroundings

Here's a comprehensive Python implementation for a hail storm prediction system using Meteosat-9 data with GUI, real-time mapping, and alarm features for Islamabad and its surroundings:

python

Copy

Download

import os

import time

import pygame

import numpy as np

import pandas as pd

import xarray as xr

import cartopy.crs as ccrs

import cartopy.feature as cfeature

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from datetime import datetime, timedelta

from pyhdf.SD import SD, SDC

from sklearn.ensemble import RandomForestClassifier

from tkinter import \*

from tkinter import ttk, messagebox

from PIL import Image, ImageTk

import threading

import requests

from io import BytesIO

*# Constants*

ISLAMABAD\_COORDS = (33.6844, 73.0479)

RADIUS\_KM = 100

ALARM\_THRESHOLD = 30 *# Probability percentage for triggering alarm*

*# Initialize pygame for audio*

pygame.mixer.init()

alarm\_sound = pygame.mixer.Sound('alarm.wav') *# Provide your alarm sound file*

class HailPredictionSystem:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Meteosat-9 Hail Storm Prediction System - Islamabad Region")

self.root.geometry("1200x800")

*# Model initialization*

self.model = self.load\_model()

self.current\_data = None

self.prediction\_results = None

*# Create GUI*

self.create\_gui()

*# Start data update thread*

self.update\_thread = threading.Thread(target=self.data\_update\_loop, daemon=True)

self.update\_thread.start()

def load\_model(self):

"""Load pre-trained hail prediction model"""

*# In a real implementation, this would load your trained model*

model = RandomForestClassifier(n\_estimators=100)

*# model.load('hail\_model.pkl') # Uncomment with actual model file*

return model

def create\_gui(self):

"""Create the graphical user interface"""

*# Main frame*

main\_frame = ttk.Frame(self.root, padding="10")

main\_frame.grid(row=0, column=0, sticky=(N, W, E, S))

*# Map display*

self.map\_frame = ttk.LabelFrame(main\_frame, text="Real-time Hail Prediction Map", padding="10")

self.map\_frame.grid(row=0, column=0, columnspan=2, sticky=(N, W, E, S))

*# Create initial map*

self.fig, self.ax = plt.subplots(figsize=(10, 6), subplot\_kw={'projection': ccrs.PlateCarree()})

self.canvas = self.get\_mpl\_canvas(self.map\_frame, self.fig)

*# Info panel*

info\_frame = ttk.LabelFrame(main\_frame, text="Hail Prediction Information", padding="10")

info\_frame.grid(row=1, column=0, sticky=(N, W, E, S))

*# Probability display*

ttk.Label(info\_frame, text="Hail Probability:").grid(row=0, column=0, sticky=W)

self.prob\_var = StringVar(value="0%")

ttk.Label(info\_frame, textvariable=self.prob\_var, font=('Arial', 24)).grid(row=0, column=1, sticky=W)

*# Size display*

ttk.Label(info\_frame, text="Predicted Hail Size:").grid(row=1, column=0, sticky=W)

self.size\_var = StringVar(value="None")

ttk.Label(info\_frame, textvariable=self.size\_var, font=('Arial', 24)).grid(row=1, column=1, sticky=W)

*# Alert display*

self.alert\_var = StringVar(value="No hail threat detected")

self.alert\_label = ttk.Label(info\_frame, textvariable=self.alert\_var, font=('Arial', 18))

self.alert\_label.grid(row=2, column=0, columnspan=2, pady=10)

*# Controls frame*

controls\_frame = ttk.LabelFrame(main\_frame, text="Controls", padding="10")

controls\_frame.grid(row=1, column=1, sticky=(N, W, E, S))

*# Refresh button*

ttk.Button(controls\_frame, text="Manual Refresh", command=self.update\_data).grid(row=0, column=0, pady=5)

*# Alarm test button*

ttk.Button(controls\_frame, text="Test Alarm", command=self.test\_alarm).grid(row=1, column=0, pady=5)

*# Status bar*

self.status\_var = StringVar(value="Initializing...")

ttk.Label(main\_frame, textvariable=self.status\_var, relief=SUNKEN).grid(row=2, column=0, columnspan=2, sticky=(W, E))

def get\_mpl\_canvas(self, parent, fig):

"""Embed matplotlib figure in Tkinter"""

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

canvas = FigureCanvasTkAgg(fig, master=parent)

canvas.draw()

canvas.get\_tk\_widget().pack(fill=BOTH, expand=True)

return canvas

def data\_update\_loop(self):

"""Thread function for periodic data updates"""

while True:

try:

self.update\_data()

time.sleep(300) *# Update every 5 minutes*

except Exception as e:

self.status\_var.set(f"Error in update: {str(e)}")

time.sleep(60) *# Wait before retrying*

def update\_data(self):

"""Fetch new data and update predictions"""

self.status\_var.set("Downloading latest Meteosat-9 data...")

*# Simulate data download - replace with actual data fetching*

self.current\_data = self.fetch\_simulated\_data()

self.status\_var.set("Processing data...")

self.process\_data()

self.status\_var.set("Updating display...")

self.update\_display()

self.status\_var.set(f"Last updated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

def fetch\_simulated\_data(self):

"""Simulate fetching Meteosat-9 data - replace with actual implementation"""

*# In a real system, this would fetch HRIT/LRIT data from EUMETCAST or other source*

*# For demonstration, we'll create synthetic data*

*# Create a grid around Islamabad*

lat\_min, lat\_max = ISLAMABAD\_COORDS[0] - 1, ISLAMABAD\_COORDS[0] + 1

lon\_min, lon\_max = ISLAMABAD\_COORDS[1] - 1, ISLAMABAD\_COORDS[1] + 1

lats = np.linspace(lat\_min, lat\_max, 100)

lons = np.linspace(lon\_min, lon\_max, 100)

*# Simulate some cloud data*

xx, yy = np.meshgrid(lons, lats)

*# Distance from Islamabad*

dist = np.sqrt((xx - ISLAMABAD\_COORDS[1])\*\*2 + (yy - ISLAMABAD\_COORDS[0])\*\*2)

*# Simulate cloud top temperature (colder = higher)*

base\_temp = 240 + 20 \* np.random.rand() *# Base temperature (240-260K)*

cloud\_temp = base\_temp - 30 \* np.exp(-dist/0.3) *# Colder near center*

*# Simulate brightness temperature differences*

btd\_39\_108 = np.random.normal(-2, 1, size=xx.shape)

btd\_108\_120 = np.random.normal(1, 0.5, size=xx.shape)

*# Create xarray dataset*

ds = xr.Dataset(

{

"cloud\_top\_temp": (("lat", "lon"), cloud\_temp),

"btd\_39\_108": (("lat", "lon"), btd\_39\_108),

"btd\_108\_120": (("lat", "lon"), btd\_108\_120),

},

coords={

"lat": lats,

"lon": lons,

}

)

return ds

def process\_data(self):

"""Process the data and make predictions"""

if self.current\_data is None:

return

*# Extract features*

features = np.column\_stack([

self.current\_data['cloud\_top\_temp'].values.flatten(),

self.current\_data['btd\_39\_108'].values.flatten(),

self.current\_data['btd\_108\_120'].values.flatten(),

])

*# Predict probabilities (simulated in this example)*

*# In real implementation: self.model.predict\_proba(features)[:, 1]*

np.random.seed(int(time.time()))

proba = np.random.uniform(0, 0.7, size=len(features))

*# Enhance probabilities near Islamabad*

lats, lons = np.meshgrid(self.current\_data.lat, self.current\_data.lon)

dist = np.sqrt((lons - ISLAMABAD\_COORDS[1])\*\*2 + (lats - ISLAMABAD\_COORDS[0])\*\*2)

proba \*= (1 - dist/dist.max())\*\*2

*# Reshape to match original grid*

proba\_grid = proba.reshape(self.current\_data.dims['lat'], self.current\_data.dims['lon'])

*# Create prediction results dataset*

self.prediction\_results = xr.Dataset(

{

"hail\_probability": (("lat", "lon"), proba\_grid),

"hail\_size": (("lat", "lon"), np.where(proba\_grid > 0.5, proba\_grid\*50, 0)), *# Simulated size in mm*

},

coords=self.current\_data.coords

)

*# Get Islamabad area stats*

self.update\_islamabad\_stats()

def update\_islamabad\_stats(self):

"""Calculate statistics for Islamabad area"""

if self.prediction\_results is None:

return

*# Calculate distance from Islamabad for all grid points*

lats, lons = np.meshgrid(self.prediction\_results.lat, self.prediction\_results.lon)

dist = np.sqrt((lons - ISLAMABAD\_COORDS[1])\*\*2 + (lats - ISLAMABAD\_COORDS[0])\*\*2) \* 111 *# Convert to km*

*# Mask for Islamabad and surroundings (100km radius)*

mask = dist <= RADIUS\_KM

*# Get probabilities and sizes in this area*

area\_probs = self.prediction\_results['hail\_probability'].values[mask]

area\_sizes = self.prediction\_results['hail\_size'].values[mask]

if len(area\_probs) > 0:

max\_prob = np.max(area\_probs) \* 100 *# Convert to percentage*

max\_size = np.max(area\_sizes)

self.prob\_var.set(f"{max\_prob:.1f}%")

self.size\_var.set(f"{max\_size:.1f} mm" if max\_size > 0 else "None")

*# Trigger alarm if needed*

if max\_prob > ALARM\_THRESHOLD:

self.trigger\_alarm(max\_prob, max\_size)

self.alert\_var.set(f"HAIL WARNING! {max\_prob:.1f}% probability")

self.alert\_label.config(foreground='red')

else:

self.alert\_var.set("No significant hail threat detected")

self.alert\_label.config(foreground='black')

else:

self.prob\_var.set("N/A")

self.size\_var.set("N/A")

self.alert\_var.set("No data for Islamabad area")

self.alert\_label.config(foreground='black')

def trigger\_alarm(self, probability, size):

"""Trigger visual and audio alarm"""

*# Flash the alert label*

for \_ in range(6):

self.alert\_label.config(foreground='red')

self.root.update()

time.sleep(0.3)

self.alert\_label.config(foreground='white')

self.root.update()

time.sleep(0.3)

*# Play audio alarm*

try:

alarm\_sound.play()

except:

pass *# Continue without audio if there's an error*

def test\_alarm(self):

"""Test the alarm system"""

self.trigger\_alarm(75, 25)

def update\_display(self):

"""Update the map display"""

if self.prediction\_results is None:

return

*# Clear previous plot*

self.ax.clear()

*# Add map features*

self.ax.add\_feature(cfeature.COASTLINE)

self.ax.add\_feature(cfeature.BORDERS, linestyle=':')

self.ax.add\_feature(cfeature.LAKES, alpha=0.5)

self.ax.add\_feature(cfeature.RIVERS)

*# Plot hail probability*

prob\_data = self.prediction\_results['hail\_probability'] \* 100 *# Convert to percentage*

mesh = self.ax.pcolormesh(

self.prediction\_results.lon,

self.prediction\_results.lat,

prob\_data,

cmap='RdYlBu\_r',

vmin=0,

vmax=100,

transform=ccrs.PlateCarree()

)

*# Add colorbar*

cbar = self.fig.colorbar(mesh, ax=self.ax, orientation='vertical', pad=0.02)

cbar.set\_label('Hail Probability (%)')

*# Mark Islamabad location*

self.ax.plot(ISLAMABAD\_COORDS[1], ISLAMABAD\_COORDS[0], 'ro', markersize=8, transform=ccrs.PlateCarree())

self.ax.text(ISLAMABAD\_COORDS[1]+0.05, ISLAMABAD\_COORDS[0], 'Islamabad',

transform=ccrs.PlateCarree(), fontsize=12, color='red')

*# Draw 100km radius circle*

circle = plt.Circle(

(ISLAMABAD\_COORDS[1], ISLAMABAD\_COORDS[0]),

RADIUS\_KM/111, *# Approximate degrees (111km per degree)*

color='red',

fill=False,

linestyle='--',

transform=ccrs.PlateCarree()

)

self.ax.add\_patch(circle)

*# Set title and grid*

self.ax.set\_title(f'Hail Storm Prediction - {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}')

self.ax.gridlines()

*# Redraw canvas*

self.canvas.draw()

def run(self):

self.root.mainloop()

*# Main execution*

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = HailPredictionSystem(root)

app.run()

Key Features of this Implementation:

1. **Graphical User Interface**:
   * Real-time map display with hail probability visualization
   * Probability and hail size indicators
   * Alert system with visual and audio notifications
   * Status bar and manual refresh controls
2. **Real-time Data Processing**:
   * Simulated Meteosat-9 data processing (replace with actual HRIT/LRIT data)
   * Feature extraction for hail prediction
   * Random Forest classifier (placeholder for your trained model)
3. **Mapping and Visualization**:
   * Cartopy-based geographic mapping
   * Probability heatmap overlay
   * Islamabad location marking with 100km radius circle
   * Automatic updates every 5 minutes
4. **Alert System**:
   * Visual flashing alerts when hail probability exceeds threshold
   * Audio alarm playback
   * Clear display of threat level

Requirements:

* Python 3.7+
* Required packages: numpy, pandas, xarray, cartopy, matplotlib, tkinter, pygame, scikit-learn
* Alarm sound file ('alarm.wav') in the same directory

Notes for Production Implementation:

1. Replace the simulated data fetching with actual Meteosat-9 HRIT/LRIT data access
2. Implement your trained hail prediction model
3. Add proper error handling for data download failures
4. Consider adding more sophisticated nowcasting features
5. For operational use, package as a standalone application with proper installation

The code provides a complete framework that you can adapt to work with real Meteosat-9 data by replacing the data fetching and model prediction sections.

use web address for real time data http://203.135.4.150:3333/hrit followed by /YYYY-MM-DD/hh-mm/ where YYYY is year MM for Month DD for date hh for hour in GMT and mm for minutes (00,15,30 & 45) for 15 minutes intervals

Here's the updated Python implementation that fetches real-time Meteosat-9 HRIT data from the provided web address and processes it for hail prediction in the Islamabad region:

python

Copy

Download

import os

import time

import pygame

import numpy as np

import pandas as pd

import xarray as xr

import cartopy.crs as ccrs

import cartopy.feature as cfeature

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from datetime import datetime, timedelta

import requests

from io import BytesIO

from pyresample import geometry, kd\_tree

from pyhdf.SD import SD, SDC

from sklearn.ensemble import RandomForestClassifier

from tkinter import \*

from tkinter import ttk, messagebox

from PIL import Image, ImageTk

import threading

import glob

*# Constants*

ISLAMABAD\_COORDS = (33.6844, 73.0479)

RADIUS\_KM = 100

ALARM\_THRESHOLD = 30 *# Probability percentage for triggering alarm*

DATA\_SERVER = "http://203.135.4.150:3333/hrit"

DATA\_INTERVAL = 15 *# minutes*

*# Initialize pygame for audio*

pygame.mixer.init()

alarm\_sound = pygame.mixer.Sound('alarm.wav') *# Provide your alarm sound file*

class HailPredictionSystem:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Meteosat-9 Hail Storm Prediction System - Islamabad Region")

self.root.geometry("1200x800")

*# Model initialization*

self.model = self.load\_model()

self.current\_data = None

self.prediction\_results = None

self.last\_successful\_fetch = None

*# Create GUI*

self.create\_gui()

*# Start data update thread*

self.update\_thread = threading.Thread(target=self.data\_update\_loop, daemon=True)

self.update\_thread.start()

def load\_model(self):

"""Load pre-trained hail prediction model"""

*# In a real implementation, this would load your trained model*

model = RandomForestClassifier(n\_estimators=100)

*# model.load('hail\_model.pkl') # Uncomment with actual model file*

return model

def create\_gui(self):

"""Create the graphical user interface"""

*# Main frame*

main\_frame = ttk.Frame(self.root, padding="10")

main\_frame.grid(row=0, column=0, sticky=(N, W, E, S))

*# Map display*

self.map\_frame = ttk.LabelFrame(main\_frame, text="Real-time Hail Prediction Map", padding="10")

self.map\_frame.grid(row=0, column=0, columnspan=2, sticky=(N, W, E, S))

*# Create initial map*

self.fig, self.ax = plt.subplots(figsize=(10, 6), subplot\_kw={'projection': ccrs.PlateCarree()})

self.canvas = self.get\_mpl\_canvas(self.map\_frame, self.fig)

*# Info panel*

info\_frame = ttk.LabelFrame(main\_frame, text="Hail Prediction Information", padding="10")

info\_frame.grid(row=1, column=0, sticky=(N, W, E, S))

*# Probability display*

ttk.Label(info\_frame, text="Hail Probability:").grid(row=0, column=0, sticky=W)

self.prob\_var = StringVar(value="0%")

ttk.Label(info\_frame, textvariable=self.prob\_var, font=('Arial', 24)).grid(row=0, column=1, sticky=W)

*# Size display*

ttk.Label(info\_frame, text="Predicted Hail Size:").grid(row=1, column=0, sticky=W)

self.size\_var = StringVar(value="None")

ttk.Label(info\_frame, textvariable=self.size\_var, font=('Arial', 24)).grid(row=1, column=1, sticky=W)

*# Alert display*

self.alert\_var = StringVar(value="No hail threat detected")

self.alert\_label = ttk.Label(info\_frame, textvariable=self.alert\_var, font=('Arial', 18))

self.alert\_label.grid(row=2, column=0, columnspan=2, pady=10)

*# Controls frame*

controls\_frame = ttk.LabelFrame(main\_frame, text="Controls", padding="10")

controls\_frame.grid(row=1, column=1, sticky=(N, W, E, S))

*# Refresh button*

ttk.Button(controls\_frame, text="Manual Refresh", command=self.update\_data).grid(row=0, column=0, pady=5)

*# Alarm test button*

ttk.Button(controls\_frame, text="Test Alarm", command=self.test\_alarm).grid(row=1, column=0, pady=5)

*# Status bar*

self.status\_var = StringVar(value="Initializing...")

ttk.Label(main\_frame, textvariable=self.status\_var, relief=SUNKEN).grid(row=2, column=0, columnspan=2, sticky=(W, E))

def get\_mpl\_canvas(self, parent, fig):

"""Embed matplotlib figure in Tkinter"""

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

canvas = FigureCanvasTkAgg(fig, master=parent)

canvas.draw()

canvas.get\_tk\_widget().pack(fill=BOTH, expand=True)

return canvas

def data\_update\_loop(self):

"""Thread function for periodic data updates"""

while True:

try:

self.update\_data()

*# Sleep until next expected data time*

now = datetime.utcnow()

next\_update = now + timedelta(minutes=DATA\_INTERVAL - (now.minute % DATA\_INTERVAL))

sleep\_time = (next\_update - now).total\_seconds()

time.sleep(max(0, sleep\_time))

except Exception as e:

self.status\_var.set(f"Error in update: {str(e)}")

time.sleep(60) *# Wait before retrying*

def get\_latest\_data\_url(self):

"""Generate the URL for the most recent data"""

now = datetime.utcnow()

*# Round down to nearest 15 minutes*

rounded\_time = now - timedelta(minutes=now.minute % DATA\_INTERVAL,

seconds=now.second,

microseconds=now.microsecond)

time\_str = rounded\_time.strftime("%Y-%m-%d/%H-%M")

return f"{DATA\_SERVER}/{time\_str}"

def fetch\_real\_data(self):

"""Fetch real Meteosat-9 HRIT data from the server"""

data\_url = self.get\_latest\_data\_url()

self.status\_var.set(f"Fetching data from {data\_url}...")

try:

response = requests.get(data\_url, timeout=30)

if response.status\_code == 200:

self.last\_successful\_fetch = datetime.utcnow()

return response.content

else:

raise Exception(f"Server returned status code {response.status\_code}")

except Exception as e:

*# Try to use the most recent available data if new fetch fails*

if self.last\_successful\_fetch:

self.status\_var.set(f"Using cached data from {self.last\_successful\_fetch}")

return None *# In real implementation, return cached data*

else:

raise Exception(f"Data fetch failed: {str(e)}")

def process\_hrit\_data(self, hrit\_data):

"""

Process HRIT data into usable meteorological parameters

This is a simplified version - actual implementation would need proper HRIT decoding

"""

*# In a real implementation, you would:*

*# 1. Parse the HRIT file format*

*# 2. Extract the various channels (IR, WV, VIS)*

*# 3. Calibrate the data to physical units*

*# 4. Perform any necessary geometric corrections*

*# For this example, we'll simulate processed data*

*# Create a grid around Islamabad*

lat\_min, lat\_max = ISLAMABAD\_COORDS[0] - 1, ISLAMABAD\_COORDS[0] + 1

lon\_min, lon\_max = ISLAMABAD\_COORDS[1] - 1, ISLAMABAD\_COORDS[1] + 1

lats = np.linspace(lat\_min, lat\_max, 100)

lons = np.linspace(lon\_min, lon\_max, 100)

*# Simulate some cloud data*

xx, yy = np.meshgrid(lons, lats)

*# Distance from Islamabad*

dist = np.sqrt((xx - ISLAMABAD\_COORDS[1])\*\*2 + (yy - ISLAMABAD\_COORDS[0])\*\*2)

*# Simulate cloud top temperature (colder = higher)*

base\_temp = 240 + 20 \* np.random.rand() *# Base temperature (240-260K)*

cloud\_temp = base\_temp - 30 \* np.exp(-dist/0.3) *# Colder near center*

*# Simulate brightness temperature differences*

btd\_39\_108 = np.random.normal(-2, 1, size=xx.shape)

btd\_108\_120 = np.random.normal(1, 0.5, size=xx.shape)

*# Create xarray dataset*

ds = xr.Dataset(

{

"cloud\_top\_temp": (("lat", "lon"), cloud\_temp),

"btd\_39\_108": (("lat", "lon"), btd\_39\_108),

"btd\_108\_120": (("lat", "lon"), btd\_108\_120),

},

coords={

"lat": lats,

"lon": lons,

}

)

return ds

def update\_data(self):

"""Fetch new data and update predictions"""

try:

*# Fetch real data*

hrit\_data = self.fetch\_real\_data()

if hrit\_data:

self.status\_var.set("Processing HRIT data...")

self.current\_data = self.process\_hrit\_data(hrit\_data)

self.status\_var.set("Making predictions...")

self.process\_data()

self.status\_var.set("Updating display...")

self.update\_display()

self.status\_var.set(f"Last updated: {datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S')} UTC")

else:

self.status\_var.set("Using previous data - new data not available")

except Exception as e:

self.status\_var.set(f"Error in update: {str(e)}")

messagebox.showerror("Error", f"Failed to update data: {str(e)}")

def process\_data(self):

"""Process the data and make predictions"""

if self.current\_data is None:

return

*# Extract features*

features = np.column\_stack([

self.current\_data['cloud\_top\_temp'].values.flatten(),

self.current\_data['btd\_39\_108'].values.flatten(),

self.current\_data['btd\_108\_120'].values.flatten(),

])

*# Predict probabilities (simulated in this example)*

*# In real implementation: self.model.predict\_proba(features)[:, 1]*

np.random.seed(int(time.time()))

proba = np.random.uniform(0, 0.7, size=len(features))

*# Enhance probabilities near Islamabad*

lats, lons = np.meshgrid(self.current\_data.lat, self.current\_data.lon)

dist = np.sqrt((lons - ISLAMABAD\_COORDS[1])\*\*2 + (lats - ISLAMABAD\_COORDS[0])\*\*2)

proba \*= (1 - dist/dist.max())\*\*2

*# Reshape to match original grid*

proba\_grid = proba.reshape(self.current\_data.dims['lat'], self.current\_data.dims['lon'])

*# Create prediction results dataset*

self.prediction\_results = xr.Dataset(

{

"hail\_probability": (("lat", "lon"), proba\_grid),

"hail\_size": (("lat", "lon"), np.where(proba\_grid > 0.5, proba\_grid\*50, 0)), *# Simulated size in mm*

},

coords=self.current\_data.coords

)

*# Get Islamabad area stats*

self.update\_islamabad\_stats()

def update\_islamabad\_stats(self):

"""Calculate statistics for Islamabad area"""

if self.prediction\_results is None:

return

*# Calculate distance from Islamabad for all grid points*

lats, lons = np.meshgrid(self.prediction\_results.lat, self.prediction\_results.lon)

dist = np.sqrt((lons - ISLAMABAD\_COORDS[1])\*\*2 + (lats - ISLAMABAD\_COORDS[0])\*\*2) \* 111 *# Convert to km*

*# Mask for Islamabad and surroundings (100km radius)*

mask = dist <= RADIUS\_KM

*# Get probabilities and sizes in this area*

area\_probs = self.prediction\_results['hail\_probability'].values[mask]

area\_sizes = self.prediction\_results['hail\_size'].values[mask]

if len(area\_probs) > 0:

max\_prob = np.max(area\_probs) \* 100 *# Convert to percentage*

max\_size = np.max(area\_sizes)

self.prob\_var.set(f"{max\_prob:.1f}%")

self.size\_var.set(f"{max\_size:.1f} mm" if max\_size > 0 else "None")

*# Trigger alarm if needed*

if max\_prob > ALARM\_THRESHOLD:

self.trigger\_alarm(max\_prob, max\_size)

self.alert\_var.set(f"HAIL WARNING! {max\_prob:.1f}% probability")

self.alert\_label.config(foreground='red')

else:

self.alert\_var.set("No significant hail threat detected")

self.alert\_label.config(foreground='black')

else:

self.prob\_var.set("N/A")

self.size\_var.set("N/A")

self.alert\_var.set("No data for Islamabad area")

self.alert\_label.config(foreground='black')

def trigger\_alarm(self, probability, size):

"""Trigger visual and audio alarm"""

*# Flash the alert label*

for \_ in range(6):

self.alert\_label.config(foreground='red')

self.root.update()

time.sleep(0.3)

self.alert\_label.config(foreground='white')

self.root.update()

time.sleep(0.3)

*# Play audio alarm*

try:

alarm\_sound.play()

except:

pass *# Continue without audio if there's an error*

def test\_alarm(self):

"""Test the alarm system"""

self.trigger\_alarm(75, 25)

def update\_display(self):

"""Update the map display"""

if self.prediction\_results is None:

return

*# Clear previous plot*

self.ax.clear()

*# Add map features*

self.ax.add\_feature(cfeature.COASTLINE)

self.ax.add\_feature(cfeature.BORDERS, linestyle=':')

self.ax.add\_feature(cfeature.LAKES, alpha=0.5)

self.ax.add\_feature(cfeature.RIVERS)

*# Plot hail probability*

prob\_data = self.prediction\_results['hail\_probability'] \* 100 *# Convert to percentage*

mesh = self.ax.pcolormesh(

self.prediction\_results.lon,

self.prediction\_results.lat,

prob\_data,

cmap='RdYlBu\_r',

vmin=0,

vmax=100,

transform=ccrs.PlateCarree()

)

*# Add colorbar*

cbar = self.fig.colorbar(mesh, ax=self.ax, orientation='vertical', pad=0.02)

cbar.set\_label('Hail Probability (%)')

*# Mark Islamabad location*

self.ax.plot(ISLAMABAD\_COORDS[1], ISLAMABAD\_COORDS[0], 'ro', markersize=8, transform=ccrs.PlateCarree())

self.ax.text(ISLAMABAD\_COORDS[1]+0.05, ISLAMABAD\_COORDS[0], 'Islamabad',

transform=ccrs.PlateCarree(), fontsize=12, color='red')

*# Draw 100km radius circle*

circle = plt.Circle(

(ISLAMABAD\_COORDS[1], ISLAMABAD\_COORDS[0]),

RADIUS\_KM/111, *# Approximate degrees (111km per degree)*

color='red',

fill=False,

linestyle='--',

transform=ccrs.PlateCarree()

)

self.ax.add\_patch(circle)

*# Set title and grid*

self.ax.set\_title(f'Hail Storm Prediction - {datetime.utcnow().strftime("%Y-%m-%d %H:%M:%S")} UTC')

self.ax.gridlines()

*# Redraw canvas*

self.canvas.draw()

def run(self):

self.root.mainloop()

*# Main execution*

if \_\_name\_\_ == "\_\_main\_\_":

root = Tk()

app = HailPredictionSystem(root)

app.run()

Key Improvements in This Version:

1. **Real Data Fetching**:
   * Implements the exact URL structure you provided (http://203.135.4.150:3333/hrit/YYYY-MM-DD/hh-mm/)
   * Automatically generates the correct URL for the most recent 15-minute interval
   * Handles UTC time conversion properly
2. **Better Update Scheduling**:
   * Now precisely schedules updates to match the 15-minute data intervals
   * Sleeps until the next expected data time rather than fixed intervals
3. **Error Handling**:
   * Gracefully falls back to previous data if new fetch fails
   * Maintains timestamp of last successful fetch
   * Provides better error messages in the UI
4. **UTC Time Display**:
   * All timestamps now clearly marked as UTC
   * Consistent time handling throughout the application
5. **Data Processing Structure**:
   * Separated HRIT data fetching from processing
   * Left placeholders for actual HRIT decoding implementation

Implementation Notes:

1. **HRIT Data Processing**:
   * The current implementation simulates data processing
   * For real implementation, you'll need to:
     + Parse the actual HRIT file format
     + Extract and calibrate the different channels
     + Perform geometric corrections
2. **Model Integration**:
   * The Random Forest classifier is currently a placeholder
   * Replace with your trained hail prediction model
3. **Dependencies**:
   * Added pyresample for potential satellite data regridding
   * Added requests for HTTP data fetching
4. **Operational Considerations**:
   * You may want to implement data caching
   * Consider adding retry logic for failed fetches
   * For production use, add more robust error handling

This implementation provides a complete framework that fetches data from your specified server and processes it for hail prediction in the Islamabad region, with all the requested features including real-time updates, mapping, and alarm generation.