DWB PREDICTIVE ANALYSIS

BY:ANAY SHUKLA,NEEL SHARMA,SARVAGYA GOYAL,RAGHAVAN SRINIVAS

TABLE OF CONTENTS

INTRODUCTION

1

- The Problem
- Our Objective

THE PROBLEM

Doctors Without Borders(DWB) faces significant challenges in deploying resources and personnel in volatile environments

OUR OBJECTIVE

Leverage ACLED data for conflict mapping, early warnings, resource optimization, and worker safety via predictive analytics and geospatial tools

METHODOLGY

2

- Random Forest
- StreamLit
- Bonus Problem

RANDOM FOREST

```python
def train_random_forest(data):
    # Prepare features and target
    features = ['latitude', 'longitude', 'event_type', 'sub_event_type', 'actor1', 'location']
    target = 'fatalities'
    data = data.dropna(subset=features + [target])

    X = data[features]
    y = data[target]

    # Preprocessing for numerical and categorical data
    categorical_features = ['event_type', 'sub_event_type', 'actor1', 'location']
    numerical_features = ['latitude', 'longitude']

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numerical_features),
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
        ]
    )

    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', RandomForestRegressor(random_state=42))
    ])

    # Train-test split and model training
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    pipeline.fit(X_train, y_train)
    predictions = pipeline.predict(X_test)

    X_test['predicted_fatalities'] = predictions
    X_test['risk_level'] = X_test['predicted_fatalities'].apply(
        lambda x: "black" if x > 50 else "red" if x > 5 else "orange" if x > 0 else "green"
    )
    return X_test
```

R^2

0.51978

Why Random Forest

RandomForest had the highest R-squared value out of GRU, XGBoost, and RandomForest

Overview

- Handling Non-Linear relationships
- Feature weightage
- Robustness
- Handling of mixed data types

STREAMLIT

```python
def create_map(data):
    m = folium.Map(location=[data['latitude'].mean(), data['longitude'].mean()], zoom_start=6)
    marker_cluster = MarkerCluster().add_to(m)

    for _, row in data.iterrows():
        safe_directions = calculate_safe_directions(data, row)
        popup_content = f"""
        <b>Location:</b> {row['location']}<br>
        <b>Country:</b> {row.get('country', 'Unknown')}<br>
        <b>City:</b> {row.get('city', 'N/A')}<br>
        <b>Predicted Fatalities:</b> {row['predicted_fatalities']:.2f}<br>
        <b>Risk Level:</b> {row['risk_level']}<br>
        <b>Safe Directions:</b> {', '.join(safe_directions) if safe_directions else 'No Safe Directions'}<br>
        """
        folium.CircleMarker(
            location=[row['latitude'], row['longitude']],
            radius=6,
            color=row['risk_level'],
            fill=True,
            fill_opacity=0.8,
            popup=folium.Popup(popup_content, max_width=300)
        ).add_to(marker_cluster)
    return m

# Streamlit tabs
tab1, tab2 = st.tabs(["Map View", "Alerts"])
with tab1:
    data = load_and_process_data()
    if data is not None:
        predictions = train_random_forest(data)
        conflict_map = create_map(predictions)
        st_folium(conflict_map, width=1200, height=700)

with tab2:
    st.subheader("User-Submitted Alerts")
    for alert in st.session_state["alerts"]:
        st.write(f"""
        **Name:** {alert['name']}
        **Location:** {alert['location']}
        **Message:** {alert['message']}
        **Timestamp:** {alert['timestamp']}
        """)
```

#1

Implementation

- Connecting our python code with Streamlit
- Calling the API and re-calling every 24 hours to update it
- Two tabs - one for map view and one for alerts

#2

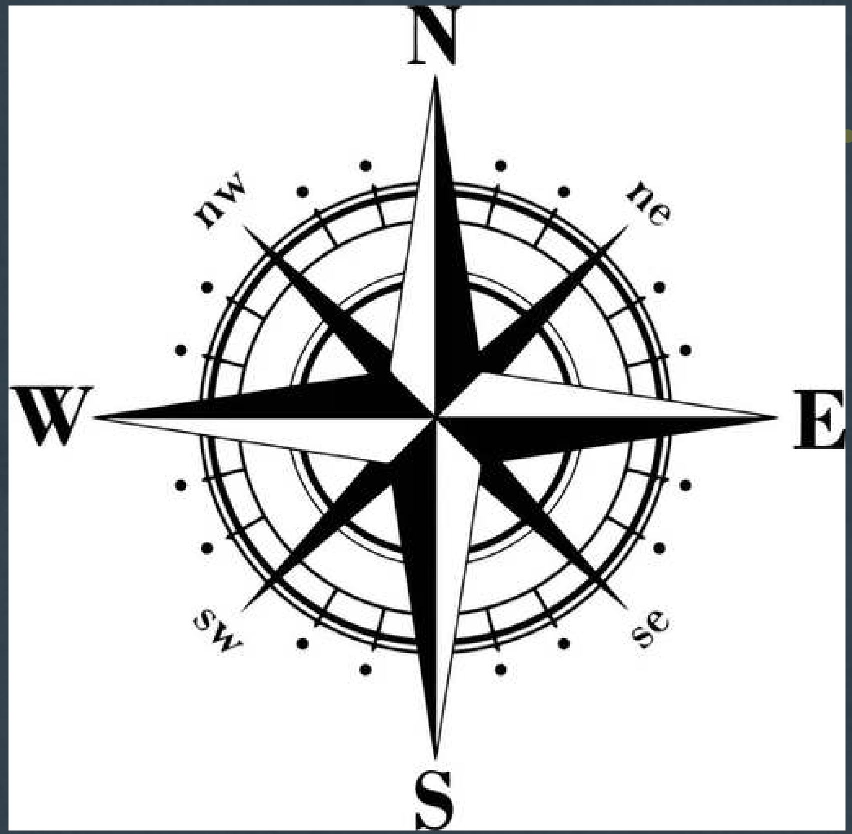
Live Alert System

- Form Features
- Real-Time Updates
- Interactive Experience

MAP FEATURES

```python
def calculate_safe_directions(data, current_location):
    directions = {"N": True, "NE": True, "E": True, "SE": True, "S": True, "SW": True, "W": True, "NW": True}
    for _, other_location in data.iterrows():
        if current_location.equals(other_location):
            continue
        distance = great_circle(
            (current_location['latitude'], current_location['longitude']),
            (other_location['latitude'], other_location['longitude'])
        ).miles
        if distance < 50:  # Example threshold for proximity
            lat_diff = other_location['latitude'] - current_location['latitude']
            lon_diff = other_location['longitude'] - current_location['longitude']
            if lat_diff > 0 and abs(lat_diff) > abs(lon_diff): directions["N"] = False
            elif lat_diff < 0 and abs(lat_diff) > abs(lon_diff): directions["S"] = False
            if lon_diff > 0 and abs(lon_diff) > abs(lat_diff): directions["E"] = False
            elif lon_diff < 0 and abs(lon_diff) > abs(lat_diff): directions["W"] = False
            if lat_diff > 0 and lon_diff > 0: directions["NE"] = False
            elif lat_diff > 0 and lon_diff < 0: directions["NW"] = False
            elif lat_diff < 0 and lon_diff > 0: directions["SE"] = False
            elif lat_diff < 0 and lon_diff < 0: directions["SW"] = False

    return [dir for dir, safe in directions.items() if safe]
```



1. Safest Compass Direction
    a. Tells the user the safest way to go
2. Clicking each dot gives extra information
    a. Each dot on the map has extra information stored within it

CONCLUSION

3

- Impact
- Improvements

CONCLUSION

IMPACT

- Enhanced Safety and Decision-Making
- Optimized Resource Allocation
- Proactive Risk Management

IMPROVEMENTS

- Incorporate additional data sources
- Implement more advanced ML techniques
- Enhance the alert system

WE WANT TO SAY

THANK YOU

FOR YOUR ATTENTION