**Data Preparation/Feature Engineering**

**1. Overview**

In creating our predictive flood model for resilient Sierra Leone communities, the data preparation and feature engineering stage is pivotal. Drawing from Open Weather's historical data (January 1979 to December 2, 2023), encompassing temperature, rainfall, longitudes, latitudes, 1-hour and 3-hour rainfall, humidity, pressure, weather description, wind, wind speed, feels-like temperature, dew point, clouds, etc., we shape the foundation for an accurate and adaptable model. This phase serves as the bedrock upon which my machine learning model will be constructed, influencing its accuracy, generalization, and interpretability. The project utilized a comprehensive historical weather data of Sierra Leone from Open Weather spanning January 1979 to December 2, 2023.

**2. Data Collection**

The dataset used in the project is sourced from OpenWeather, specifically containing historical weather data for Sierra Leone spanning from 1979 to 2023. OpenWeather is a reputable platform that provides access to weather data from various locations around the world. The dataset was made available after specifying the specified time range from January 1, 1979, to December 4, 2023, and making payment via an email link sent from Open Weather. No preprocessing step was taken during the data collection as I considered all columns relevant to give me an understanding of the various columns that will be pertinent to building my flood prediction model.

**Data Source:** [https://history.openweathermap.org/storage/52b96426702dd7eed9a4507cd5c11615.csv /](https://history.openweathermap.org/storage/52b96426702dd7eed9a4507cd5c11615.csv%20/) <https://home.openweathermap.org/marketplace/my_orders>

**3. Data Cleaning**

Steps taken to clean the dataset include:

1. Identified columns with missing values using the isnull().sum() methods to determine number of rows in each column that has missing values
2. Handled missing values for columns; rain\_1h and rain\_3h, wind\_gust by using the fillna() method and setting the value to zero (0). This indicates the unavailability of rain. Missing values in visibility column was handled using .fillna() method as well filling missing values with the median of the column. See code snippet below:

flood\_training\_data["rain\_1h"] = flood\_training\_data['rain\_1h'].fillna(0)

flood\_training\_data["rain\_3h"] = flood\_training\_data['rain\_3h'].fillna(0)

flood\_training\_data["wind\_gust"] = flood\_training\_data['wind\_gust'].fillna(0)

flood\_training\_data["visibility"].fillna(flood\_training\_data["visibility"].median(), inplace=True)

1. Summary statistics using the .describe() was used to identify outliers and outliers.
2. Time series transformation was done to convert the dt column in my dataset to datetime format.
3. One hot encoding was done to encode categorical data if any.
4. Removed irrelevant features in the dataset that do not contribute meaningful information to the analysis.
5. Document all steps taken during the data cleaning process using comments.
6. Created a new feature ‘rain’ by combining rain\_1h and rain\_3h to be one.

# Create a new feature 'rain' by combining 'rain\_1h' and 'rain\_3h'

flood\_training\_data["rain"] = flood\_training\_data["rain\_1h"] + flood\_training\_data["rain\_3h"]

1. Created a target variable for feature selection using rainfall and setting the threshold to 4.0 and more to account for flood and less for no flood. See code snippet below:

# Create binary target variable

# Set a rainfall threshold to be considered flood or not

threshold = 4.0

flood\_training\_data['flood\_occurred'] = np.where(flood\_training\_data['rain'] >= threshold, 1, 0)

1. Converted the target variable into a pandas series:

#converts the y into a Pandas Series

y = pd.Series(y)

print(y.isnull().sum())

1. Removed single-dimensional entries in the target variable using the .squeeze() function:

y = y.squeeze()

**4. Exploratory Data Analysis (EDA)**

1. Computed summary statistics for key variables, including measures of central tendency and dispersion using the .describe() function.
2. Created a correlation matrix and visualized it using a heatmap.

A screenshot of a computer screen

Description automatically generated

1. Performed pair plotting to gain insights into the structure, relationships, and patterns within the dataset.

A group of blue dots

Description automatically generated

**5. Feature Engineering**

1. Created new features for rain by combining rain\_1h and rain\_3h.

# Create a new feature 'rain' by combining 'rain\_1h' and 'rain\_3h'

flood\_training\_data["rain"] = flood\_training\_data["rain\_1h"] + flood\_training\_data["rain\_3h"]

1. Created a target variable using rain as the most important feature since it’s fain that causes flooding. This was done by setting a threshold of flooding to 4.0, i.e, rainfall of 4.0 or more was considered to have the potential to cause flooding.

# Create binary target variable

# Set a rainfall threshold to be considered flood or not

threshold = 4.0

flood\_training\_data['flood\_occurred'] = np.where(flood\_training\_data['rain'] >= threshold, 1, 0)

1. Features were selected for the X variable and target variable y respectively. The features selected are outlined as follows:

features = ['temp', 'dew\_point', 'feels\_like', 'temp\_min', 'temp\_max', 'wind\_speed', 'wind\_gust', 'rain', 'clouds\_all']

X = flood\_training\_data[features]

y = flood\_training\_data['flood\_occurred']

the decision was based on my judgement that these features are important weather metrics that determine the likely occurrence of a flood event.

1. Encoding of categorical variables were done using sci-kit learn LabelEncoder as shown

# Encoding Categorical Variables

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

**6. Data Transformation**

1. Categorical data were transformed by encoding using sci-kit learn LabelEncoder as shown

# Encoding Categorical Variables

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

1. Converted the target y to a pandas dataframe and used the isnull() sum() function to check missing values.

#converts the y into a Pandas Series

y = pd.Series(y)

print(y.isnull().sum())

1. Removed single dimensional entries in the y variable.

#remove single-dimensional entries

y = y.squeeze()

**Model Exploration**

**1. Model Selection**

1. ***RandomForestClassifier:***

**Rationale:**

* **Ensemble Robustness:** Combining multiple decision trees mitigates the risk of overfitting and improves generalization to unseen data, crucial for predicting floods with diverse weather patterns.

**Strengths:**

* Robust performance due to the combination of multiple decision trees.
* Can capture non-linear relationships in the data.
* Provides a ranking of feature importance.

**Weaknesses:**

* Training can be computationally expensive, especially with a large number of decision trees.
* The ensemble nature may make it less interpretable compared to simpler models.

1. ***Logistic Regression:***

**Rationale:**

* **Interpretability:** Logistic Regression provides a simple, interpretable model, allowing for a clear understanding of the relationship between weather features and flood likelihood.
* **Baseline Efficiency:** Serves as a baseline model for quick comparisons with more complex models, establishing a reference point for performance.

**Strengths:**

* Easy to interpret and understand.
* Fast training and prediction times, particularly beneficial for large datasets.

**Weaknesses:**

* Assumes a linear relationship, potentially limiting its ability to capture complex non-linearities.
* May not perform as well when the relationship between features and flood likelihood is highly non-linear.

1. ***KNeighborsClassifier*:**

**Rationale:**

* **Local Adaptability:** Effective for capturing localized patterns in weather conditions that influence floods.
* **Non-Parametric Flexibility:** No assumptions about the underlying data distribution, providing flexibility in adapting to different patterns.

**Strengths:**

* Adapts well to local patterns.
* Non-parametric and makes minimal assumptions about the data distribution.

**Weaknesses:**

* Prediction time can be computationally expensive, especially with large datasets.
* Sensitive to noisy data and outliers.

1. ***SVC (Support Vector Classifier):***

**Rationale:**

* **High-Dimensional Capability:** Effective in high-dimensional spaces, suitable for weather datasets with numerous features.
* **Versatility with Kernels:** Different kernel functions provide flexibility in capturing complex relationships in the data.

**Strengths:**

* Works well in datasets with a large number of features.
* Can capture complex relationships using various kernel functions.

**Weaknesses:**

* Training can be computationally intensive, especially with large datasets.
* Performance can be sensitive to the choice of hyperparameters and the kernel function.

1. ***DecisionTreeClassifier:***

**Rationale:**

* Decision Trees offer interpretability, enabling a clear understanding of the decision-making process. The model's ability to capture non-linear relationships aligns with the complex nature of weather-flood interactions.
* Provides a simple and transparent structure, aiding in the identification of key decision points.

**Strengths:**

* Easy to interpret and visualize.
* Can capture non-linear relationships in the data.
* Transparent decision-making process.

**Weaknesses:**

* Prone to overfitting, especially when the tree is deep and captures noise in the data.
* Small variations in the data can lead to different tree structures, making the model less stable.

**2. Model Training**

The following steps were taken for the model training:

1. **Data Splitting:**

* **train\_test\_split**: The dataset was split into training and testing sets using the **train\_test\_split** function from scikit-learn. 80% of the data was used for training (**X\_train**, **y\_train**), and 20% for testing (**X\_test**, **y\_test**).

1. **Model Selection and Initialization:**

* A dictionary **models** was created, containing instances of the selected models with their respective hyperparameters.
  + - **RandomForestClassifier**: Default hyperparameters with **random\_state=None**.
    - **LogisticRegression**: Maximum iterations set to 1000, **random\_state=None**.
    - **KNeighborsClassifier**: Default hyperparameters.
    - **SVC**: Default hyperparameters with **random\_state=None**.
    - **DecisionTreeClassifier**: Default hyperparameters with **random\_state=None**.

1. **Cross-Validation:**

* A loop iterates over each model, performing 5-fold cross-validation using **cross\_val\_score**. The mean and standard deviation of accuracy scores are printed for each model.
* The purpose of cross-validation is to assess each model's performance on different subsets of the training data, helping to identify potential overfitting or underfitting.

1. **Learning Curve Visualization:**

* Another loop iterates over each model to generate learning curves.
* **learning\_curve** is used to obtain training and validation accuracy scores across different training set sizes. These curves are then plotted to visualize the models' performance as the training size increases.

A graph of a graph showing different types of training

Description automatically generated with medium confidence

1. **Model Training and Evaluation:**

* For each model, the code fits the model to the training data (**model.fit(X\_train, y\_train)**), makes predictions on the test data, and evaluates the model's performance using accuracy, classification report, and confusion matrix metrics.

1. **Results Printing:**

* The accuracy, classification report, and a separator are printed for each model.

**Evaluation Metrics:**

* Accuracy is used as the primary metric for evaluating model performance.
* Classification reports are printed for a more detailed understanding of precision, recall, F1-score, and support for each class.

**Learning Curves:**

* Learning curves visualize how the models' performance changes as the size of the training set increases. The shaded areas around the curves represent the standard deviation.

**3. Model Evaluation**

For evaluating the mode, accuracy, precision, recall and F1-score were the performance metrics used. Below is the result of the model training:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

**RandomForest**

* **Accuracy:** 100.0%
* **Precision:** 1.00 (both for class 0 and class 1)
* **Recall:** 1.00 (both for class 0 and class 1)
* **F1-score:** 1.00 (both for class 0 and class 1)

The RandomForest model achieves perfect accuracy, precision, recall, and F1-score, indicating flawless performance. It correctly predicts all instances of both classes (0 and 1). The model shows no false positives or false negatives. However, an accuracy of 100% might mean the model is overfitting.

**LogisticRegression**

* **Accuracy:** 99.9886%
* **Precision:** 1.00 (both for class 0 and class 1)
* **Recall:** 1.00 (both for class 0 and class 1)
* **F1-score:** 1.00 (both for class 0 and class 1)

The LogisticRegression model also performs exceptionally well, with high accuracy and perfect precision, recall, and F1-score for both classes. It correctly predicts nearly all instances.

**KNeighbors**

* **Accuracy:** 99.8451%
* **Precision:** 1.00 for class 0, 0.97 for class 1
* **Recall:** 1.00 for class 0, 0.93 for class 1
* **F1-score:** 1.00 for class 0, 0.95 for class 1

The KNeighbors model achieves high accuracy and perfect precision, recall, and F1-score for class 0. For class 1, while precision is still high, recall and F1-score are slightly lower, indicating some false negatives.

**SVC (Support Vector Classifier)**

* **Accuracy:** 98.6794%
* **Precision:** 0.99 for class 0, 1.00 for class 1
* **Recall:** 1.00 for class 0, 0.19 for class 1
* **F1-score:** 0.99 for class 0, 0.32 for class 1

The SVC model demonstrates high accuracy but faces challenges in recall and F1-score for class 1. It has high precision for class 1, suggesting that when it predicts class 1, it is often correct. However, its recall for class 1 is relatively low, indicating it misses some true positives.

**DecisionTree**

* **Accuracy:** 100.0%
* **Precision:** 1.00 (both for class 0 and class 1)
* **Recall:** 1.00 (both for class 0 and class 1)
* **F1-score:** 1.00 (both for class 0 and class 1)

Similar to the RandomForest model, the DecisionTree model achieves perfect accuracy, precision, recall, and F1-score, indicating flawless performance without any misclassifications.

**4. Code Implementation**

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, learning\_curve

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

from sklearn.feature\_selection import SelectKBest, f\_classif

# Load data

flood\_training\_data = pd.read\_csv("Sierra\_Leone-Weather\_Data.csv")

print(flood\_training\_data.head())

A screenshot of a computer

Description automatically generated

# Data Preprocessing

# Handling Missing Values

flood\_training\_data["rain\_1h"] = flood\_training\_data['rain\_1h'].fillna(0)

flood\_training\_data["rain\_3h"] = flood\_training\_data['rain\_3h'].fillna(0)

# Create a new feature 'rain' by combining 'rain\_1h' and 'rain\_3h'

flood\_training\_data["rain"] = flood\_training\_data["rain\_1h"] + flood\_training\_data["rain\_3h"]

index\_value = 5893

column\_values = flood\_training\_data.loc[index\_value, 'rain']

print(column\_values)

# Create binary target variable

# Set a rainfall threshold to be considered flood or not

threshold = 4.0

flood\_training\_data['flood\_occurred'] = np.where(flood\_training\_data['rain'] >= threshold, 1, 0)

# check for missing or null values

print(flood\_training\_data.isnull().sum())

flood\_training\_data["wind\_gust"] = flood\_training\_data['wind\_gust'].fillna(0)

#.fillna(flood\_training\_data["wind\_gust"].median(), inplace=True)

# Feature Engineering

features = ['temp', 'dew\_point', 'feels\_like', 'temp\_min', 'temp\_max', 'wind\_speed', 'wind\_gust', 'rain', 'clouds\_all']

X = flood\_training\_data[features]

y = flood\_training\_data['flood\_occurred']

# Encoding Categorical Variables

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# Exploratory Data Analysis (EDA)

# Descriptive Statistics

print("Descriptive Statistics:\n", X.describe())

# Visualization - pairplot

sns.pairplot(X)

plt.show()

# Correlation Heatmap

correlation\_matrix = X.corr()

plt.figure(figsize=(16,9))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Feature Correlation Heatmap")

plt.show()

#converts the y into a Pandas Series

y = pd.Series(y)

print(y.isnull().sum())

#remove single-dimensional entries

y = y.squeeze()

# Model Selection

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=None)

# Choose models

models = {

'RandomForest': RandomForestClassifier(random\_state=None),

'LogisticRegression': LogisticRegression(max\_iter=1000, random\_state=None),

'KNeighbors': KNeighborsClassifier(),

'SVC': SVC(random\_state=None),

'DecisionTree': DecisionTreeClassifier(random\_state=None)

}

# Cross-validation

for name, model in models.items():

scores = cross\_val\_score(model, X, y, cv=5, scoring='accuracy')

print(f"{name} Cross-Validation Accuracy: {np.mean(scores):.4f} (std: {np.std(scores):.4f})")

# Learning Curve

for name, model in models.items():

train\_sizes, train\_scores, test\_scores = learning\_curve(model, X, y, cv=5, scoring='accuracy', train\_sizes=np.linspace(0.1, 1.0, 10))

train\_mean = np.mean(train\_scores, axis=1)

train\_std = np.std(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

test\_std = np.std(test\_scores, axis=1)

# plt.figure(figsize=(16,9))

plt.plot(train\_sizes, train\_mean, label=f"{name} Training")

plt.fill\_between(train\_sizes, train\_mean - train\_std, train\_mean + train\_std, alpha=0.15)

plt.plot(train\_sizes, test\_mean, label=f"{name} Validation")

plt.fill\_between(train\_sizes, test\_mean - test\_std, test\_mean + test\_std, alpha=0.15)

plt.title("Learning Curves")

plt.xlabel("Training Size")

plt.ylabel("Accuracy")

plt.legend()

plt.show()

A graph of a graph showing different types of training

Description automatically generated with medium confidence

for model\_name, model in models.items():

# Train the model

model.fit(X\_train, y\_train)

# Make predictions

predictions = model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, predictions)

report = classification\_report(y\_test, predictions, zero\_division=1)

# Print the results

print(f"Model: {model\_name}")

print(f"Accuracy: {accuracy\*100}")

print("Classification Report:")

print(report)

print("\n" + "="\*50 + "\n")

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated