**Machine Learning Project Documentation**

**Model Refinement**

**1. Overview**

The model refinement phase is a critical stage in the development of a machine learning model, focused on enhancing its performance, generalization, and robustness. In this phase, the initial model, often based on default settings or preliminary hyperparameters, undergoes a series of iterative improvements and optimizations. The primary goal is to achieve a model that not only fits the training data well but also demonstrates strong predictive capabilities on unseen data.

**2. Model Evaluation**

**Summary of Initial Model Evaluation Results:**

The initial model evaluation results indicate outstanding performance across all models, with high accuracy, precision, recall, and F1-scores. Here's a summarized overview of the key findings:

***RandomForest:***

* **Accuracy:** 100.0%
* **Precision, Recall, F1-score (both classes 0 and 1):** Perfect scores.
* **Summary:** The RandomForest model demonstrates flawless performance, correctly predicting all instances in both classes.

***LogisticRegression:***

* **Accuracy:** 99.99%
* **Precision, Recall, F1-score (both classes 0 and 1):** Perfect scores.
* **Summary:** LogisticRegression exhibits exceptional performance with near-perfect scores, indicating accurate predictions for both classes.

***KNeighbors:***

* **Accuracy:** 99.85%
* **Precision, Recall, F1-score (class 0):** Perfect scores.
* **Precision, Recall, F1-score (class 1):** Slightly lower recall and F1-score.
* **Summary:** While KNeighbors achieves high accuracy, there is room for improvement in correctly identifying instances of class 1, as indicated by the lower recall and F1-score for class 1.

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

* **Accuracy:** 98.68%
* **Precision, Recall, F1-score (class 0):** High scores.
* **Precision, Recall, F1-score (class 1):** Lower recall and F1-score.
* **Summary:** SVC performs well overall but faces challenges in correctly identifying instances of class 1, as indicated by the lower recall and F1-score for class 1.

***DecisionTree:***

* **Accuracy:** 100.0%
* **Precision, Recall, F1-score (both classes 0 and 1):** Perfect scores.
* **Summary:** Similar to RandomForest, DecisionTree demonstrates flawless performance, accurately predicting all instances in both classes.

**Areas for Improvement:**

***KNeighbors - Class 1 Prediction:***

* While KNeighbors achieves high accuracy, there is room for improvement in correctly identifying instances of class 1. Strategies such as hyperparameter tuning or exploring alternative models may enhance its performance for class 1 predictions.

***SVC - Class 1 Prediction:***

* SVC shows challenges in correctly identifying instances of class 1, with lower recall and F1-score for class 1. Further refinement, such as adjusting hyperparameters or exploring different kernels, may improve its ability to predict class 1 instances.

***Overall Consideration:***

* Given the exceptional performance of RandomForest and DecisionTree, the refinement phase may focus on optimizing hyperparameters and exploring ensemble techniques to further enhance model robustness.

***Visualizations:***

* **Learning Curves:** The learning curves generated during the model exploration phase may provide insights into the models' convergence and potential areas for improvement, especially for models with slight discrepancies in performance.

**Next Steps:**

* The refinement phase should involve systematic exploration of hyperparameters, feature engineering, and potentially experimenting with ensemble methods to further improve model performance.

**3. Refinement Techniques**

The refinement phase involved a comprehensive set of techniques to enhance the machine learning model's performance. These techniques encompassed adjustments to hyperparameters, the introduction of new features, the exploration of additional algorithms, and addressing class imbalance through oversampling.

1. ***New Feature Addition:***

New features 'dt', 'timezone', 'lat', 'lon', 'visibility', 'pressure', 'humidity' were added to the dataset. These features likely provided the models with additional information, potentially improving their ability to capture patterns related to flood prediction.

1. ***One-Hot Encoding:***

The 'timezone' column was one-hot encoded using the **pd.get\_dummies** function. One-hot encoding is a common technique to represent categorical variables, and it was applied to ensure compatibility with machine learning algorithms that require numerical input.

1. ***Introduction of New Algorithms:***

Three new algorithms, 'NeuralNetwork' using MLPClassifier and ensemble method GradientBoosting, XGBoost, wereintroduced to expand the range of models explored. The inclusion of these algorithms provides diversity and allows for the identification of a model that may better capture complex relationships within the data.

1. ***Oversampling with SMOTE:***

To address the issue of class imbalance, Synthetic Minority Over-sampling Technique (SMOTE) was employed. SMOTE generates synthetic samples for the minority class, thereby balancing the class distribution. This is crucial for improving the model's ability to predict the minority class accurately and prevent biases towards the majority class.

1. ***Hyperparameter Tuning:***

Hyperparameters of the models, specifically the LogisticRegression model, were tuned. Adjustments to parameters such as the random state and max iterations were made to optimize the model's convergence and performance.

1. ***Model Selection:***

To reduce computational time and focus on the most promising models, the refinement phase narrowed down the model selection to LogisticRegression and 'NeuralNetwork' (MLPClassifier). This decision was likely based on the performance observed during the exploration phase, considering factors such as accuracy, precision, recall, and F1-score.

1. ***Feature Engineering:***

The addition of new features and the subsequent exploration of their impact on model performance represents a form of feature engineering. This process aimed to enhance the representational power of the model by providing it with more relevant information.

**4. Cross-Validation**

The cross-validation strategy was modified during the model refinement phase. In the initial model, a standard k-fold cross-validation approach with k=5 was used. However, in the refined model, the strategy was changed to Stratified K-Fold cross-validation with k=5.

1. The refined model introduces Stratified K-Fold cross-validation, which is particularly useful when dealing with imbalanced datasets. Stratified K-Fold ensures that each fold maintains the same class distribution as the original dataset. This is crucial for maintaining representative subsets, especially if the target classes are imbalanced.
2. In the refined model, a StandardScaler is applied to numeric features before cross-validation. This standardization ensures that numeric features have similar scales, which can be important
3. In the refined model, non-numeric columns are dropped before applying the StandardScaler. This is a practical approach as some scaling methods may not be applicable to non-numeric data, and dropping these columns ensures that only numeric features are scaled for certain algorithms that are sensitive to the scale of input features.
4. The **shuffle=True** parameter in Stratified K-Fold introduces randomization within each fold, potentially avoiding issues related to data ordering. This can be beneficial, especially if the dataset has some inherent ordering that might impact the model's performance.

**5. HyperParameter Tuning**

Hyperparameter tuning was performed using GridSearchCV on the logistic regression. The regularization parameter **C** for Logistic Regression is tuned. The values tested are **[0.001, 0.01, 0.1, 1, 10, 100]**. The regularization parameter controls the inverse of the regularization strength. Tuning this parameter helps find the optimal balance between fitting the training data well and avoiding overfitting.

The best hyperparameter found is **{'C': 100}**. This suggests that a stronger regularization (lower values of **C**) might not be necessary, and the model benefits from allowing more flexibility in the decision boundaries.

The model with the best hyperparameters achieved a very high accuracy of **0.9999**. This suggests that the hyperparameter tuning process was successful in finding a set of parameters that maximizes model performance on the test set.



**6. Feature Selection**

Feature selection was performed by introducing new features ('dt', 'timezone', 'lat', 'lon', 'visibility', 'pressure', 'humidity') in the refined model. The impact of this feature selection on the model's performance can be assessed by comparing the classification reports and accuracy scores.

**Initial Model:**

***RandomForest, LogisticRegression, KNeighbors, DecisionTree:***

* Achieved perfect accuracy of 100% across all models.
* Indicative of potential overfitting, especially when accuracy is high.

***SVC*:**

* Shows a slightly lower accuracy of 98.68%.
* Imbalances in class 1 reflected in lower recall and F1-score for class 1.

**Refined Model:**

***Logistic Regression:***

* Accuracy slightly decreased to 99.94%.
* Classification report shows varying performance for class 0 and class 1, with precision, recall, and F1-score dropping for class 0. However, the model still performs exceptionally well.

***NeuralNetwork:***

* Accuracy 99.90%.
* Classification report indicates high performance for both classes, with minor reductions in precision, recall, and F1-score for class 0.

***Observations and Considerations:***

* The new features introduced during the refinement phase have not significantly impacted the performance of the models. The models still achieve high accuracy, indicating that the new features might not have provided substantial additional information.
* The decrease in accuracy could indicate that the refinement process mitigated potential overfitting observed in the initial model. The refined models are likely to better generalize to unseen data.
* The refined models demonstrate stability, maintaining high accuracy and performance metrics even with the introduction of new features.
* It's essential to consider the potential impact of class imbalance on model evaluation. While accuracy is high, the performance metrics for the minority class (class 1) should be carefully examined, especially in scenarios where imbalances exist.

Top of Form

**Test Submission**

**1. Overview**

The test submission phase involves preparing the machine learning model for deployment or evaluation on a test dataset. Here's an overview of the steps taken during this phase.

1. Choose the final machine learning model based on the refined and evaluated models. I considered the model that demonstrates the best performance on the validation set and aligns with the project's objectives.
2. Feature scaling and transformations that were applied during the model training and refinement phases, were applied to the features in the test dataset. This consistency is crucial for the model to make accurate predictions on new data.

**2. Data Preparation for Testing**

The test dataset was generated using random weather data for different cities or towns in Sierra Leone.

1. Utilized the Faker library to generate random weather data for various weather-related parameters such as temperature, visibility, dew point, feels-like temperature, pressure, humidity, wind speed, wind gust, rain, and cloud coverage.
2. A function (**generate\_weather\_data**) was created to generate 100 data points for each city.
3. Iterated over a list of cities in Sierra Leone, and for each city, generated random weather data using the function created in step 1.
4. Saved the generated weather data for each city as a separate CSV file ({city}\_weather\_test\_data.csv).
5. Read the CSV files for each city into separate DataFrames (data1 to data7).
6. Added a 'city' column to each DataFrame to indicate the city or town to which the weather data belongs.
7. Concatenated all DataFrames into one (all\_data) to create a unified test dataset containing weather data for multiple cities.
8. Dropped the 'city' column from the combined DataFrame, as it was used for organizational purposes and is not a feature for model training or testing.
9. Saved the combined test dataset as a CSV file (testing\_data.csv).
10. Read the saved test dataset back into a DataFrame (test\_data).
11. Replaced the 'GMT' values in the 'timezone' column with 0 to standardize the time zone representation.
12. Created a new binary column ('flood') based on the condition that if the 'rain' value is greater than or equal to 4.0, set it to 1 (indicating potential flooding), otherwise set it to 0.
13. Selected the relevant features ('timezone', 'lat', 'lon', 'temp', 'visibility', 'dew\_point', 'feels\_like', 'temp\_min', 'temp\_max', 'pressure', 'humidity', 'wind\_speed', 'wind\_gust', 'rain', 'clouds\_all') for testing.
14. Created the feature matrix (X\_test) and the target variable (y\_test\_data) for testing based on the selected features.

**3. Model Application**

The trained models were applied to the test dataset using the following code snippet:

# Loop through each model

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

# Make predictions on the test set

predictions = model.predict(X\_test)

# Evaluate the model

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

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

# Print the results

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

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

print("Test Classification Report:")

print(report)

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

* The code iterates through each trained model (models.items()), where model\_name represents the name of the model, and model is the corresponding trained machine learning model.
* For each model, predictions are generated on the test set (**X\_test**) using the **predict** method.
* The accuracy of the model is calculated using the accuracy\_score function, comparing the predicted values (predictions) with the actual values (y\_test\_data).
* The classification report, including precision, recall, and F1-score, is generated using the classification\_report function.

**4. Test Metrics**

Here are the metrics used to evaluate the model's performance on the test dataset, compared with the training and refinement metrics:

***Test Dataset Metrics:***

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Training Dataset Metrics:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* The LogisticRegression model performs exceptionally well on the validation dataset, achieving high precision, recall, and f1-score for class 1. However, on the test dataset, there is a notable drop in precision for class 1 (flood), resulting in a decrease in the overall f1-score. The macro avg and weighted avg metrics also show a decline in performance in the test dataset compared to the validation dataset.
* The NeuralNetwork model also demonstrates strong performance on the validation dataset, with high precision, recall, and f1-score for class 1. On the test dataset, precision for class 1 remains high, but there is a significant drop in recall, resulting in a lower f1-score. Both macro avg and weighted avg metrics indicate a decrease in performance on the test dataset compared to the validation dataset, with the weighted avg experiencing a more pronounced decline.

**5. Model Deployment**

The trained model will be deployed into a real-world setting through a Python Flask web application.

* A Flask web application is developed in Python to serve as the deployment platform.
* The application includes an HTML form for users to select a city.
* When a user selects a city and submits the form, the Flask application makes an API call to OpenWeatherMap using their API endpoint.
* The OpenWeatherMap API provides real-time weather data for the selected city.
* The raw weather data obtained from the API is preprocessed within the Flask application.
* Data preprocessing involves formatting the raw data into the required input features for the machine learning model.
* The pre-trained machine learning model, serialized using pickle, is loaded into the Flask application.
* The preprocessed weather data is passed into the model to make predictions about the likelihood of flooding.
* The prediction result is returned to the Flask application and displayed to the user through their web browser.
* The application provides information about the likelihood of flooding based on the real-time weather data.

The Flask application acts as an intermediary between the user and the machine learning model. It fetches real-time weather data, processes it, and feeds it into the model for predictions. The results are then presented to the user through a simple web interface. This deployment allows users to access flood predictions for different cities in real-time through their web browsers.

**6. 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.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.neural\_network import MLPClassifier

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

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import GridSearchCV

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())

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

# 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)

rainfall = 'rain\_1h'

rainfall\_data = flood\_training\_data[rainfall]

print(rainfall\_data.head(20))

# 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)

print(flood\_training\_data['flood\_occurred'][5892])

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

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

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

print(flood\_training\_data['visibility'].isnull())

print(flood\_training\_data['visibility'].head())

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

# Feature Engineering

features = ['dt', 'timezone', 'lat', 'lon', 'temp', 'visibility', 'dew\_point', 'feels\_like', 'temp\_min', 'temp\_max', 'pressure', 'humidity','wind\_speed', 'wind\_gust', 'rain', 'clouds\_all']

X = flood\_training\_data[features]

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

# Time Series Transformation

# Convert the 'dt' column to datetime

X['dt'] = pd.to\_datetime(flood\_training\_data['dt'])

X = X.set\_index('dt')

# One-hot encode 'timezone' column

X\_encoded = pd.get\_dummies(X, columns=['timezone'])

from sklearn.preprocessing import LabelEncoder

# Encoding Categorical Variables

label\_encoder = LabelEncoder()

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

# Exploratory Data Analysis (EDA)

# Descriptive Statistics

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

# Visualization

sns.pairplot(X)

plt.show()

# Correlation Heatmap

# Select only numeric columns excluding 'lon' and 'lat'

numeric\_columns = X\_encoded.drop(['lon', 'lat'], axis=1).select\_dtypes(include=['float64', 'int64'])

# Calculate correlation matrix

corr\_matrix = numeric\_columns.corr()

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

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

plt.title("Feature Correlation Heatmap")

plt.show()

y = pd.Series(y)

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

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=42)

# Select models

models = {

# 'RandomForest': RandomForestClassifier(),

'LogisticRegression': LogisticRegression(max\_iter=1500),

# 'KNeighbors': KNeighborsClassifier(),

# 'SVC': SVC(),

# 'DecisionTree': DecisionTreeClassifier(),

# 'GradientBoosting': GradientBoostingClassifier(),

'NeuralNetwork': MLPClassifier(),

# 'XGBoost': XGBClassifier()

}

# Cross-validation

from sklearn.model\_selection import StratifiedKFold

# Drop non-numeric columns

X\_numeric = X.select\_dtypes(include=['float64', 'int64'])

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_numeric)

for name, model in models.items():

# StratifiedKFold for cross-validation

cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

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

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

# Perform oversampling using SMOTE to handle imbalance

from imblearn.over\_sampling import SMOTE, RandomOverSampler

smote = SMOTE(random\_state=42)

X\_resampled\_smote, y\_resampled\_smote = smote.fit\_resample(X\_train, y\_train)

# Perform oversampling using RandomOverSampler

resampler = RandomOverSampler(random\_state=42)

X\_resampled\_random, y\_resampled\_random = resampler.fit\_resample(X\_train, y\_train)

# Choose one of the resampled datasets

X\_resampled = X\_resampled\_smote

y\_train\_resampled = y\_resampled\_smote

X\_resampled = X\_resampled\_smote

y\_train\_resampled = y\_resampled\_smote

# Feature Engineering and Regularization

# Hyperparameter tuning using GridSearchCV

param\_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

grid\_search = GridSearchCV(LogisticRegression(max\_iter=1000), param\_grid, cv=5)

grid\_search.fit(X\_resampled, y\_train\_resampled)

# Print the best hyperparameters

print("Best hyperparameters:", grid\_search.best\_params\_)

# Evaluate the model with the best hyperparameters

best\_logistic\_regression\_model = grid\_search.best\_estimator\_

best\_accuracy = best\_logistic\_regression\_model.score(X\_test, y\_test)

print("Accuracy with best hyperparameters:", best\_accuracy)

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

# Train the model

model.fit(X\_resampled\_smote, y\_resampled\_smote)

# 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")

from sklearn.metrics import classification\_report, roc\_auc\_score

y\_pred = best\_logistic\_regression\_model.predict(X\_test)

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

print("AUC-ROC Score:", roc\_auc\_score(y\_test, best\_logistic\_regression\_model.predict\_proba(X\_test)[:, 1]))

# Use pickle for model serialization

import pickle

# Save the trained model to a file using pickle

with open('final\_refined\_prediction\_model.pkl', 'wb') as model\_file:

pickle.dump(model, model\_file, protocol=pickle.HIGHEST\_PROTOCOL)

from joblib import dump, load

# Save the model

dump(model, 'final\_refined\_prediction\_model.joblib')

**Conclusion**

In the model refinement phase, the initial evaluation showcased exceptional performance, with RandomForest and DecisionTree models demonstrating flawless accuracy and precision. LogisticRegression exhibited high accuracy, but a slight decline during refinement, particularly in metrics related to class 1 predictions. KNeighbors achieved overall high accuracy but revealed room for improvement in correctly identifying instances of class 1. Support Vector Classifier (SVC) performed well, yet faced challenges in accurately predicting class 1 instances. The refinement strategies involved the addition of new features, one-hot encoding, introduction of new algorithms, oversampling with SMOTE to address class imbalance, hyperparameter tuning, and focused model selection. The refined models maintained stability, demonstrating high accuracy even with the introduction of new features. The chosen models, LogisticRegression and NeuralNetwork, underwent cross-validation using a modified approach to better handle imbalanced datasets. The test submission phase involved preparing the model for deployment by applying consistent feature scaling and transformations to the test dataset. This comprehensive refinement process aimed to enhance the model's generalization, with a particular focus on improving class 1 predictions where challenges were identified.

**References**

1. Mosavi, P. Ozturk, and K.-W. Chau, “Flood Prediction Using Machine Learning Models: Literature Review.”
2. “Reflections on Sierra Leone’s mudslide disaster - BBC News.” Accessed: Nov. 20, 2023. [Online]. Available: <https://www.bbc.com/news/world-africa-40973539>
3. H. Moon, S. Yoon, and Y. Moon, “Urban flood forecasting using a hybrid modeling approach based on a deep learning technique,” Journal of Hydroinformatics, vol. 25, no. 2, pp. 593–610, Mar. 2023, doi: 10.2166/hydro.2023.203.
4. “OpenWeather Marketplace.” Accessed: Dec. 09, 2023. [Online]. Available: <https://home.openweathermap.org/marketplace/my_orders>
5. N. Gauhar, S. Das, and K. S. Moury, “Prediction of Flood in Bangladesh using k-Nearest Neighbors Algorithm,” International Conference on Robotics, Electrical and Signal Processing Techniques, pp. 357–361, 2021, doi: 10.1109/ICREST51555.2021.9331199.
6. D. K. Behera, M. Das, S. Swetanisha, and P. K. Sethy, “Hybrid model for movie recommendation system using content K-nearest neighbors and restricted Boltzmann machine,” Indonesian Journal of Electrical Engineering and Computer Science, vol. 23, no. 1, pp. 445–452, Jul. 2021, doi: 10.11591/IJEECS.V23.I1.PP445-452.
7. A. Tella, M. R. Ul Mustafa, A. O. Balogun, C. J. Okolie, I. B. Yamusa, and M. B. Ibrahim, “SPATIAL PREDICTION OF FLOOD IN KUALA LUMPUR CITY OF MALAYSIA USING LOGISTIC REGRESSION,” in International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives, International Society for Photogrammetry and Remote Sensing, Feb. 2023, pp. 363–369. doi: 10.5194/isprs-archives-XLVIII-4-W6-2022-363-2023.
8. A. Sahoo, S. Samantaray, and D. K. Ghose, “Prediction of Flood in Barak River using Hybrid Machine Learning Approaches: A Case Study,” Journal of the Geological Society of India, vol. 97, no. 2, pp. 186–198, Feb. 2021, doi: 10.1007/S12594-021-1650-1/METRICS.
9. I. M. Hayder et al., “An Intelligent Early Flood Forecasting and Prediction Leveraging Machine and Deep Learning Algorithms with Advanced Alert System,” Processes 2023, Vol. 11, Page 481, vol. 11, no. 2, p. 481, Feb. 2023, doi: 10.3390/PR11020481.
10. “Reflections on Sierra Leone’s mudslide disaster - BBC News.” Accessed: Nov. 20, 2023. [Online]. Available: <https://www.bbc.com/news/world-africa-40973539>
11. “Sierra Leone: Floods - Emergency Plan of Action (EPoA) DREF n° MDRSL008 Final Report - Sierra Leone | ReliefWeb.” Accessed: Nov. 26, 2023. [Online]. Available: <https://reliefweb.int/report/sierra-leone/sierra-leone-floods-emergency-plan-action-epoa-dref-n-mdrsl008-final-report>
12. “Stats SL - Statistics Sierra Leone Hands Over Final Census Results To President Bio.” Accessed: Nov. 26, 2023. [Online]. Available: <https://www.statistics.sl/index.php/statistics-sierra-leone-hands-over-final-census-results-to-president-bio.html>
13. “Sierra Leone: Post-Mudslides and Floods Needs Assessment and Recovery Framework | GFDRR.” Accessed: Nov. 26, 2023. [Online]. Available: <https://www.gfdrr.org/en/sierra-leone-post-mudslides-and-floods-needs-assessment-and-recovery-framework>
14. “MDRSL008-Sierra Leone Floods-DREF Final Report.”
15. “SDG AI Lab Data Science Project Framework (Capstone Project).pptx.” Accessed: Nov. 26, 2023. [Online]. Available: <https://undp.sharepoint.com/:p:/r/sites/FrontierTechLeadersCommunity/_layouts/15/Doc.aspx?sourcedoc=%7B870D3A35-8B5A-4F52-8726-F9D6CBE9714C%7D&file=SDG%20AI%20Lab%20Data%20Science%20Project%20Framework%20(Capstone%20Project).pptx&action=edit&mobileredirect=true>
16. “Monthly Rainfall Index and Flood Probability.” Accessed: Nov. 21, 2023. [Online]. Available: <https://www.kaggle.com/datasets/mukulthakur177/kerela-flood/data>
17. A. Mosavi, P. Ozturk, and K.-W. Chau, “Flood Prediction Using Machine Learning Models: Literature Review.”
18. H. Moon, S. Yoon, and Y. Moon, “Urban flood forecasting using a hybrid modeling approach based on a deep learning technique,” Journal of Hydroinformatics, vol. 25, no. 2, pp. 593–610, Mar. 2023, doi: 10.2166/hydro.2023.203
19. A. Paul and P. Das, “Flood Prediction Model using Artificial Neural Network,” International Journal of Computer Applications Technology and Research, vol. 3, no. 7, pp. 473–478, 2014, [Online]. Available: [www.ijcat.com](http://www.ijcat.com)