**Capstone Project Concept Note and**

**Machine Learning Project Documentation**

**Model Refinement**

**Project Title: EpiGuard: Malaria Early Warning**

Team Members

1. Sumaiya Sultana
2. Mustafa Ahmed Afeef
3. Ayan Hassan Olad
4. Stacy-Kelvin Adaisky Irutingabo

**1. Overview**

The model refinement phase focuses on improving the performance of the initial machine learning model by employing various techniques such as hyperparameter tuning, feature selection, and model evaluation. This phase is crucial for enhancing the predictive accuracy and generalizability of the model to ensure it performs well on unseen data.

**2. Model Evaluation**

The initial model evaluation indicated the following results for different models:

Linear Regression: MSE = 1,368,159,633,698.757, R² = 0.6974

Ridge Regression: MSE = 1,412,691,806,093.4924, R² = 0.6876

Lasso Regression: MSE = 1,360,282,390,890.6316, R² = 0.6992

Decision Tree: MSE = 1,189,501,166,161.0059, R² = 0.7369

Random Forest: MSE = 403,483,089,543.00665, R² = 0.9108

Gradient Boosting: MSE = 322,286,471,689.22784, R² = 0.9287

Support Vector Regressor: MSE = 4,900,999,756,915.647, R² = -0.0839

XGBoost: MSE = 349,631,157,391.89075, R² = 0.9227

The Gradient Boosting model showed the best performance with the lowest MSE of approximately 322,286,471,689.22784 and an R² score of 0.9287. The Random Forest model also performed well. Key metrics highlighted the need for further refinement to reduce prediction errors.

**3. Refinement Techniques**

During the refinement phase, we utilized several techniques to enhance model performance:

* **Hyperparameter Tuning:** Adjusting hyperparameters to optimize model performance.
* **Trying Different Algorithms:** Multiple algorithms were evaluated, including Linear Regression, Ridge Regression, Lasso Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, Support Vector Regressor, and XGBoost.
* **Data Preprocessing:** Imputing missing values, scaling features, and encoding categorical variables.

#### **4. Hyperparameter Tuning**

In the refinement phase, we focused on tuning the hyperparameters of the Gradient Boosting Regressor. The goal was to identify the optimal set of hyperparameters that would yield the best performance. The hyperparameters tuned included:

* learning\_rate: Adjusted to control the contribution of each tree.
* max\_depth: Set to limit the depth of each tree.
* min\_samples\_split: Defined the minimum number of samples required to split an internal node.
* n\_estimators: Determined the number of boosting stages to be run.

The GridSearchCV method was used to perform an exhaustive search over specified parameter values for the estimator. The best hyperparameters identified were:

* learning\_rate: 0.1
* max\_depth: 4
* min\_samples\_split: 10
* n\_estimators: 200

These hyperparameters resulted in the following model performance:

* **Mean Squared Error:** 274,642,433,985.88464
* **R² Score:** 0.9392596176786658

The hyperparameter tuning led to a significant improvement in model performance, as reflected by the increased R² score and reduced Mean Squared Error compared to the initial model evaluation.

#### **5. Feature Selection**

Feature selection was applied by removing less important features such as Rural\_population, Urban\_population, avg\_wind\_speed\_kmh, min\_temp\_c, and max\_temp\_c. This step helped in reducing the complexity of the model and potentially overfitting, leading to more robust performance.

### **Test Submission**

#### **1. Overview**

The test submission phase involves preparing the final model for deployment or evaluation on a test dataset. This phase ensures that the model is ready for real-world application and performs well on new, unseen data.

#### **2. Data Preparation for Testing**

The test dataset was prepared by ensuring it was preprocessed in the same manner as the training data. This included imputing missing values, scaling numerical features, and encoding categorical variables consistently.

**3. Model Application**

The trained Gradient Boosting model was applied to the test dataset. The following code snippet demonstrates the application:

# Apply the preprocessor to the test data

X\_test\_preprocessed = preprocessor.transform(X\_test)

# Make predictions with the trained model

test\_predictions = gb\_model.predict(X\_test\_preprocessed)

# Evaluate the model on the test data

test\_mse = mean\_squared\_error(y\_test, test\_predictions)

test\_r2 = r2\_score(y\_test, test\_predictions)

#### **4. Model Deployment**

For deployment, the model was integrated with a system that allows users to input a location and receive predictions for potential malaria cases. This integration involves setting up an API endpoint and ensuring the model is accessible for real-time predictions.

#### **5. Code Implementation**

The following code snippets highlight key sections of the model refinement and test submission phases:

**Model Refinement Code:**

# Grid Search for Hyperparameter Tuning

param\_grid = {

'learning\_rate': [0.05, 0.1, 0.15],

'max\_depth': [3, 4, 5],

'min\_samples\_split': [7, 10, 15],

'n\_estimators': [100, 150, 200]

}

grid\_search = GridSearchCV(GradientBoostingRegressor(random\_state=100), param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

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

best\_params = grid\_search.best\_params\_

print(f'Best Gradient Boosting Params: {best\_params}')

**Model Application Code:**

# Apply the model to test data

test\_predictions = gb\_model.predict(X\_test\_preprocessed)

test\_mse = mean\_squared\_error(y\_test, test\_predictions)

test\_r2 = r2\_score(y\_test, test\_predictions)

print(f'Test MSE: {test\_mse}')

print(f'Test R²: {test\_r2}')

### **Conclusion**

The model refinement and test submission phases led to a significant improvement in the model's performance. The Gradient Boosting model, after hyperparameter tuning, showed excellent predictive accuracy with an MSE of 274,642,433,985.88464 and an R² score of 0.9392596176786658. The model was successfully deployed and integrated for real-time predictions. Challenges included handling missing data and optimizing hyperparameters, but the final model achieved robust performance suitable for practical applications.