CODE BOOKLET

MICROCLIMATE DATA ANALYSIS AT CULTURAL HERITAGE SITES USING THE RANDOM FOREST AND XGBOOST ALGORITHMS

Academic Session: 2023/2024

Student: Iman Aidi Elham Bin Hairul Nizam (A20EC5006)

Program: Bachelor of Computer Science (Software Engineering)

Supervisor: Assoc. Prof. Dr. Mohd Shahizan Bin Othman

GitHub Repository: https://github.com/drshahizan/undergraduate-project/tree/main/PSM2/eddy

Email: imanaidielham@gmail.com

Summary:

This document holds all the coding for the final year project Microclimate Data Analysis at Cultural Heritage Sites Using the Random Forest and XGBoost Algorithms, which include all modules and functions for the algorithms. The research predicts microclimate variables specifically at a heritage site in Johor Bahru and Melaka such as temperature, rainfall, wind speed and humidity using the Random Forest and XGBoost algorithms.

Table of Contents

1. Project Hardware and Software List 3
2. Modules List and Function Explanation 4
3. Code for Importing Libraries 5
4. Code for Reading and Preprocessing Data 5
5. Code for Feature Engineering 6
6. Code for Checking Data Quality 7
7. Code for Training RandomForest Model 7
8. Code for Training XGBoost Model 8
9. Code for Predicting Values 8
10. Code for Evaluating Predictions 8
11. Code for Cross-Validation Evaluation 9
12. Code for Creating Plots 9
13. Code for Creating Accuracy Tables 11
14. Code for Main Function for Workflow Execution 11

Project Hardware and Software List

HARDWARE	VERSION	USAGE
PC	MSI-GL63RC Intel Core i7 8th Gen	Used to execute and train the machine learning models.

SOFTWARE	VERSION	USAGE
Spyder	4.2.5	Used to develop the coding and and run experiments.
Python	3.8.8	The programming language used to write scripts for data preprocessing, model training, evaluation, and plotting.
Notebook++	8.6.9	Used for code editing, documentation, and managing project files.

Module List and Function Explanation

MODULE	FUNCTION
Importing Libraries	Import necessary libraries for data manipulation, model training, evaluation, and visualization.
Reading and Preprocessing Data	Reads CSV data, maps months to numerical values, and excludes a specified year if provided.
Feature Engineering	Creates lagged features, rolling averages, seasonality features, interaction terms, and applies log transformation to rainfall. Filters data for the specified variable.
Checking Data Quality	Checks for NaN or infinity values in the dataset.
Training RandomForest Model	Trains a RandomForest model with hyperparameter tuning using GridSearchCV.
Training XGBoost Model	Trains an XGBoost model with hyperparameter tuning using GridSearchCV.
Predicting Values	Predicts values using the trained model.
Evaluating Predictions	Evaluates predictions against actual data and calculates MAE, RMSE, and R-squared metrics.
Cross-Validation Evaluation	Evaluates the model using cross-validation and averages the metrics across folds.
Creating Plots	Creates and saves plots for predicted vs. actual values for each variable, location, and model.
Creating Accuracy Tables	Creates and saves heatmaps of accuracy results for each location.
Main Function for Workflow Execution	Executes the complete workflow, including reading data, training models, generating predictions, and saving plots and tables for each location.

Code for Importing Libraries

```
# Import necessary libraries for data manipulation, model training, evaluation, and visualization
import pandas as pd
import numpy as np
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV, TimeSeriesSplit
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

Code for Reading and Preprocessing Data

```
# Reads CSV data, maps months to numerical values, and excludes a specified year if provided
def read_and_preprocess_data(data_file, exclude_year=None):
    data = pd.read_csv(data_file)
    month_mapping = {
        'January': 1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June': 6,
        'July': 7, 'August': 8, 'September': 9, 'October': 10, 'November': 11, 'December': 12,
        'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6, 'Jul': 7, 'Aug': 8,
        'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12
    }
    data['Month_Num'] = data['Month'].map(month_mapping)
    if exclude_year is not None:
```

```
    data = data[data['Year'] != exclude_year]
  return data
```

Code for Feature Engineering

```
# Creates lagged features, rolling averages, seasonality features, interaction terms, and applies log transformation to rainfall
def prepare_data(data, variable):
  for var in ['Temperature', 'Humidity', 'Rainfall', 'Wind Speed']:
    data[f'{var}_lag1'] = data.groupby('Year')[var].shift(1)
    data[f'{var}_lag2'] = data.groupby('Year')[var].shift(2)

  for var in ['Temperature', 'Humidity', 'Rainfall', 'Wind Speed']:
    data[f'{var}_rolling3'] = data.groupby('Year')[var].rolling(window=3, min_periods=1).mean().reset_index(0, drop=True)

  data['month_sin'] = np.sin(2 * np.pi * data['Month_Num']/12)
  data['month_cos'] = np.cos(2 * np.pi * data['Month_Num']/12)
  data['temp_humid'] = data['Temperature'] * data['Humidity']
  data['Rainfall_log'] = np.log1p(data['Rainfall'])

  y = data[variable] if variable != 'Rainfall' else data['Rainfall_log']
  X = data.drop(['Year', 'Month', 'Temperature', 'Humidity', 'Rainfall', 'Wind Speed'], axis=1)

  if variable == 'Rainfall':
    X['Rainfall_lag1'] = np.log1p(X['Rainfall_lag1'])
    X['Rainfall_lag2'] = np.log1p(X['Rainfall_lag2'])
    X['Rainfall_rolling3'] = np.log1p(X['Rainfall_rolling3'])
```

```
    X = X.replace([np.inf, -np.inf], np.nan)
    X = X.fillna(X.mean())
    y = y.fillna(y.mean())

    return X, y
```

Code for Checking Data Quality

```
# Checks for NaN or infinity values in the dataset
def check_data_quality(X, y):
    print("Checking for NaN or infinity values:")
    print("X contains NaN:", X.isna().any().any())
    print("y contains NaN:", y.isna().any())
    print("X contains infinity:", np.isinf(X).any().any())
    print("y contains infinity:", np.isinf(y).any())
```

Code for Training RandomForest Model

```
# Trains a RandomForest model with hyperparameter tuning using GridSearchCV
def train_model(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20, 30],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
    rf = RandomForestRegressor(random_state=42)
    grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2,
scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)
```

```
    print("Best parameters for Random Forest:", grid_search.best_params_)
    return grid_search.best_estimator_
```

Code for Training XGBoost Model

```
# Trains an XGBoost model with hyperparameter tuning using GridSearchCV
def train_xgboost_model(X_train, y_train):
    param_grid = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.01, 0.1, 0.3],
        'max_depth': [3, 5, 7],
        'min_child_weight': [1, 3, 5]
    }
    xgb = XGBRegressor(random_state=42)
    grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
    grid_search.fit(X_train, y_train)
    print("Best parameters for XGBoost:", grid_search.best_params_)
    return grid_search.best_estimator_
```

Code for Predicting Values

```
# Predicts values using the trained model
def predict(model, X):
    return model.predict(X)
```

Code for Evaluating Predictions

```
# Evaluates predictions against actual data and calculates MAE, RMSE, and R-squared metrics
def evaluate(predictions, actual_data):
```

```
    mae = mean_absolute_error(actual_data, predictions)
    rmse = np.sqrt(mean_squared_error(actual_data, predictions))
    r2 = r2_score(actual_data, predictions)
    metrics = {'MAE': mae, 'RMSE': rmse, 'R-squared': r2}
    return metrics
```

Code for Cross-Validation Evaluation

```
# Evaluates the model using cross-validation and averages the metrics across folds
def evaluate_with_cv(X, y, model, n_splits=5):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    metrics = {'MAE': [], 'RMSE': [], 'R-squared': []}
    for train_index, test_index in tscv.split(X):
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        model.fit(X_train, y_train)
        predictions = model.predict(X_test)
        fold_metrics = evaluate(predictions, y_test)
        for key in metrics:
            metrics[key].append(fold_metrics[key])
    return {key: np.mean(values) for key, values in metrics.items()}
```

Code for Creating Plots

```
# Creates and saves plots for predicted vs. actual values for each variable, location, and model
def create_separate_plots(predictions, actual_data, months, variables, location, models, save_dir,
metrics):
    for variable in variables:
        for model in models:
            plt.figure(figsize=(10, 6))
```

```
        plt.plot(months, actual_data[variable], marker='o', linestyle='-', color='b', label='Actual')
        plt.plot(months, predictions[f'{variable}_{model}'], marker='o', linestyle='--', color='r', label='Predicted')
        plt.xlabel('Month')
        plt.ylabel(variable)
        if variable == 'Rainfall':
            plt.title(f'{variable} - {location} - {model} (Log-transformed)')
        else:
            plt.title(f'{variable} - {location} - {model}')
        plt.xticks(range(1, 13), [str(month) for month in range(1, 13)], rotation=45)
        plt.legend()
        plt.grid(True)
        metrics_text = f"MAE: {metrics[f'{variable}_{model}']['MAE']:.2f}\n" \
                       f"RMSE: {metrics[f'{variable}_{model}']['RMSE']:.2f}\n" \
                       f"R-squared: {metrics[f'{variable}_{model}']['R-squared']:.2f}"
        plt.text(0.95, 0.05, metrics_text, transform=plt.gca().transAxes, bbox=dict(facecolor='white', alpha=0.8), fontsize=8, verticalalignment='bottom', horizontalalignment='right')
        plt.tight_layout()
        plot_filename = os.path.join(save_dir, f'{variable}_{location}_{model}.png')
        plt.savefig(plot_filename, dpi=300, bbox_inches='tight')
        plt.close()
        print(f"Plot for {variable} - {location} - {model} has been saved to '{plot_filename}'")
```

Code for Creating Accuracy Tables

```
# Creates and saves heatmaps of accuracy results for each location
def create_accuracy_tables(accuracy_results, save_dir):
    for location, metrics in accuracy_results.items():
        metrics_df = pd.DataFrame(metrics)
        plt.figure(figsize=(12, 8))
        sns.heatmap(metrics_df, annot=True, cmap='YlGnBu', fmt='.2f')
        plt.title(f'Model Evaluation Metrics for {location}')
        plt.tight_layout()
        table_filename = os.path.join(save_dir, f'accuracy_table_{location}.png')
        plt.savefig(table_filename, dpi=300, bbox_inches='tight')
        plt.close()
        print(f"Accuracy table for {location} has been saved to '{table_filename}'")
```

Code for Main Function for Workflow Execution

```
# Executes the complete workflow, including reading data, training models, generating predictions, and saving plots and tables for each location
def main_workflow(data_files, locations, variable, exclude_year, save_dir):
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)
    accuracy_results = {}
    for location, data_file in zip(locations, data_files):
        data = read_and_preprocess_data(data_file, exclude_year)
        X, y = prepare_data(data, variable)
        check_data_quality(X, y)
        tscv = TimeSeriesSplit(n_splits=5)
        metrics_dict = {}
```

```
        for train_index, test_index in tscv.split(X):
            X_train, X_test = X.iloc[train_index], X.iloc[test_index]
            y_train, y_test = y.iloc[train_index], y.iloc[test_index]
            rf_model = train_model(X_train, y_train)
            xgb_model = train_xgboost_model(X_train, y_train)
            rf_predictions = predict(rf_model, X_test)
            xgb_predictions = predict(xgb_model, X_test)
            rf_metrics = evaluate(rf_predictions, y_test)
            xgb_metrics = evaluate(xgb_predictions, y_test)
            for key in rf_metrics:
                metrics_dict.setdefault(f'RandomForest_{key}', []).append(rf_metrics[key])
                metrics_dict.setdefault(f'XGBoost_{key}', []).append(xgb_metrics[key])
        accuracy_results[location] = {key: np.mean(values) for key, values in metrics_dict.items()}
        create_separate_plots({'RandomForest': rf_predictions, 'XGBoost': xgb_predictions}, y_test,
data['Month_Num'], [variable], location, ['RandomForest', 'XGBoost'], save_dir,
accuracy_results[location])
    create_accuracy_tables(accuracy_results, save_dir)
    print(f"Workflow completed and results saved in '{save_dir}'")
```

CODE BOOKLET

SCHOOL OF COMPUTING

2024