Supplementary material

Article

Global Spatial Suitability Mapping of Wind and Solar Systems Using an Explainable AI-Based Approach

Mourtadha Sarhan Sachit [1,2], Helmi Zulhaidi Mohd Shafri [1,*], Ahmad Fikri Abdullah [3], Azmin Shakrine Mohd Rafie [4], and Mohamed Barakat A. Gibril [1,5]

[1] Department of Civil Engineering and Geospatial Information Science Research Center (GISRC), Faculty of Engineering, Universiti Putra Malaysia (UPM), Serdang 43400, Malaysia; gs58663@student.upm.edu.my; gs57386@student.upm.edu.my
[2] Department of Civil Engineering, College of Engineering, University of Thi-Qar, Nasiriyah 64001, Iraq
[3] Institut Antarabangsa Akuakultur dan Sains Akuatik (I-AQUAS), Universiti Putra Malaysia (UPM), Si Rusa 71050, Malaysia; ahmadfikri@upm.edu.my
[4] Department of Aerospace Engineering, Faculty of Engineering, Universiti Putra Malaysia (UPM), Serdang 43400, Malaysia; shakrine@upm.edu.my
[5] GIS & Remote Sensing Center, Research Institute of Sciences and Engineering, University of Sharjah, P.O. Box 27272, Sharjah, United Arab Emirates
* Correspondence: helmi@upm.edu.my; Tel.: +60-3-97696459

The programming codes implemented in this research work using Python are as follows:

- **Import dependencies**

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn import metrics
from sklearn.calibration import CalibratedClassifierCV
from sklearn.calibration import calibration_curve
```

- **Data manipulation**

```
# Reading the data
data_file = r"dataset.csv"

df = pd.read_csv(data_file)
df = df.reindex(np.random.permutation(df.index))
#df.tail()

# Defining features and target names
feature_cols = []
target_name = ""
X = df.loc[:, feature_cols]
y = df[target_name]

# Data normalization
X1=X
```

Supplementary material

```
X = np.asarray(X)
preprocessing.StandardScaler().fit(X)

# Splitting the data into training, validation and testing (70, 15, 15)
train_ratio = 0.70
validation_ratio = 0.15
test_ratio = 0.15
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1 -
train_ratio,random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test,
test_size=test_ratio/(test_ratio + validation_ratio))
print(len(X_train), len(X_val), len(X_test))
```

- **Random forest classifier**

```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_1 = rf_model.predict(X_test)

print("Precision:", metrics.precision_score(y_test, y_pred_1))
print("accuracy on test dataset: {}".format(metrics.accuracy_score(y_test,
y_pred_1)))
print("confusion_matrix: {}".format(metrics.confusion_matrix(y_test, y_pred_1)))
print("kappa coefficient: {}".format(metrics.cohen_kappa_score(y_test, y_pred_1)))
y_pred_1_prob = rf_model.predict_proba(X_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_1_prob[:,1])
roc_auc = metrics.auc(fpr, tpr)
print("Area Under the curve: {}".format(roc_auc))

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

- **Model Calibration**

```
figure(figsize=(8, 6), dpi=300)

#Uncalibrated
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_val_predict_proba = rf_model.predict_proba(X_val)[:, 1]
fraction_of_positives, mean_predicted_value = calibration_curve(y_val,
y_val_predict_proba, n_bins=10)

# plot perfectly calibrated
plt.plot([0, 1], [0, 1], linestyle='--', color='gray',label='Perfectly calibrated' )
# plot calibrated reliability
```

```
plt.plot(mean_predicted_value, fraction_of_positives, marker='.',
label='Uncalibrated')

# Calibration classifier
calibrated=CalibratedClassifierCV(rf_model, method='isotonic', cv=5)
calibrated.fit(X_train, y_train) # fit calibration model
y_value_predict_proba_cal = calibrated.predict_proba(X_val)[:, 1]
fraction_of_positives_cal, mean_predicted_value_cal=calibration_curve(y_val,
y_value_predict_proba_cal, n_bins=10)#, normalize=True)
plt.plot(mean_predicted_value_cal, fraction_of_positives_cal, marker='.',
color='red', label='Calibrated model')
plt.rcParams.update({'font.size':12})
plt.grid(linestyle='-', linewidth=0.2)
plt.xlabel("Predicted probability")
plt.ylabel("True probability")
plt.legend(loc = 'lower right')
plt.ylim([0, 1])
plt.xlim([0, 1])
plt.savefig("Solar_calibration.jpg", bbox_inches='tight', dpi=600)

### Testing the accuracy based on the testing datsaet after model calibration
y_pred_cal = calibrated.predict(X_test)
print("Precision:", metrics.precision_score(y_test, y_pred_cal))
print("accuracy on test dataset: {}".format(metrics.accuracy_score(y_test,
y_pred_cal)))
print("confusion_matrix: {}".format(metrics.confusion_matrix(y_test, y_pred_cal)))

print("kappa coefficient: {}".format(metrics.cohen_kappa_score(y_test, y_pred_cal)))
y_pred_cal_prob = calibrated.predict_proba(X_test)
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_cal_prob[:,1])
roc_auc2 = metrics.auc(fpr, tpr)
print("Area Under the curve: {}".format(roc_auc2))
```

- **EXPLainable AI**

```
import shap # package used to calculate Shap values
# load JS visualization code to notebook
shap.initjs()

import shap # package used to calculate Shap values
shap.initjs()
# Create object that can calculate shap values
explainer = shap.TreeExplainer(rf_model)
# Calculate Shap values
shap_values = explainer.shap_values(X_test)

#Summary_plot
plt.rcParams["font.family"] = "Segoe UI"
plt.rcParams.update({'text.color': "red",
                     'axes.labelcolor': "black"})
feature_cols = []
shap.summary_plot(shap_values[0], X_test,feature_names=feature_cols, plot_type="bar",
color='#E46C0A',show=False)
plt.savefig("Summary_plot_1.png",dpi=300, bbox_inches='tight')
import matplotlib.pyplot as plt
plt.rcParams["font.family"] = "Segoe UI"
plt.rcParams.update({'text.color': "red",
                     'axes.labelcolor': "black"})
shap.summary_plot(shap_values[0], X_test,feature_names=feature_cols,cmap='rainbow',
show=False)
#plt.savefig("Summary_plot_2.png",dpi=300, bbox_inches='tight')

# Force Plot
plt.rcParams.update({'font.size': 18})
f = pl.gcf()

choosen_instance = X_test[317] # select instances
shap_values = explainer.shap_values(choosen_instance)
#shap.initjs()
shap.force_plot(explainer.expected_value[0],
shap_values[0],choosen_instance,
feature_names=feature_cols,matplotlib=True,figsize=(20, 5),
text_rotation=45,show=False, )

#plt.savefig("Force_Plot_sample_317.jpg", bbox_inches='tight', dpi=600)

# Dependance plot
plt.rcParams.update({'font.size': 20})
f = pl.gcf()

Feature= 'name'
interaction_index="Name"

shap.dependence_plot(Feature,shap_values[1], X_test,feature_names=feature_cols,
dot_size=14,alpha=1,
show=False,interaction_index=interaction_index,axis_color='black',
color='black',x_jitter=0,xmin=-4, xmax=31, ymin=-0.32, ymax=0.30)

#plt.title(f"{Feature }dependence plot")
plt.ylabel(f"SHAP value for the {Feature} feature")
plt.savefig("Dependance_plot.jpg", bbox_inches='tight', dpi=600)
```

- **RF Wieghts**

```
importances = rf_model.feature_importances_
forest_importances = pd.Series(importances, index=feature_cols)
std = np.std([tree.feature_importances_ for tree in rf_model.estimators_], axis=0)
fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```

- **Prediction**

```
Data2 = "Data2.csv"

pred_1_df = pd.read_csv(Data2)
pred_1_df.tail()

feature_cols = [ ]
target_name = "need_station"
```

```
pred_1_input_data = pred_1_df.loc[:, feature_cols]
data_set_1 = pred_1_input_data.to_numpy()
preprocessing.StandardScaler().fit(data_set_1)

predictions_1 = []
for _row in data_set_1:
predictions_1.append(random_forst_predict(_row))
pred_1_df.insert(loc=len(pred_1_df.columns),column='need_solar_station',value=predict
ion_1)

#save the Predictions
Outputname='Name.csv'
pred_1_df.to_csv(Outputname)
files.download(Outputname)
```