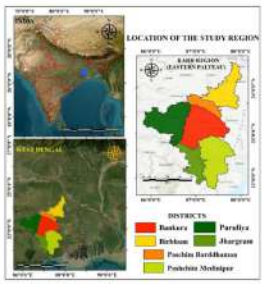
Discover Applied Sciences

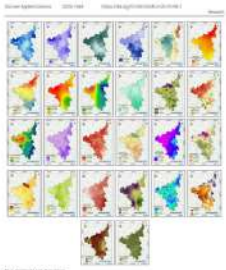
Research

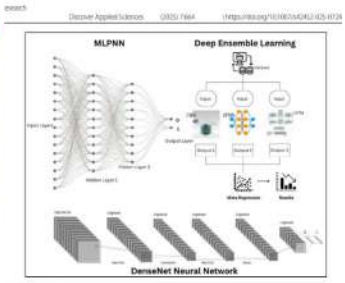
Analysing agricultural distress in the eastern plateau of West Bengal's Rarh Region: integrating hybrid deep ensemble and GIS-based soft computing

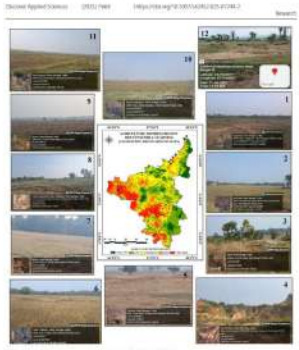
Gopal Chowdhury[1] · Ashis Kumar Saha[1]

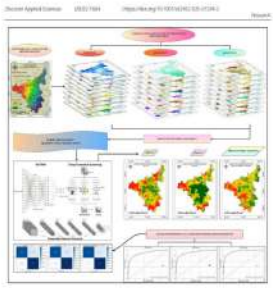
Received: 20 March 2025 / Accepted: 2 June 2025
Published online: 18 June 2025

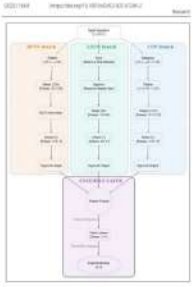


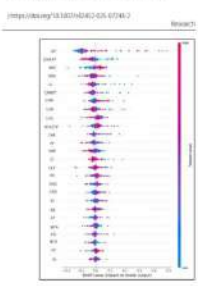












```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Conv1D, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K
import shap

warnings.filterwarnings("ignore")

# Enable eager execution
import tensorflow as tf
tf.compat.v1.enable_eager_execution()

# ----------------------
# 1. LOAD AND PREPROCESS DATA
# ----------------------

# Replace with your actual data file path
data = pd.read_csv(r'D:\1ST_OBJECTIVE_PHD\LITERATURE\Cleaned_data\DATA_3.csv')

# Split features and target
X = data.drop('FINAL_INV', axis=1)
y = data['FINAL_INV'].values
feature_names = X.columns.tolist()

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X.values, y, test_size=0.2, random_state=42)

# Normalize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape for CNN and LSTM (3D input)
X_train_scaled_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_scaled_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)

def create_cnn_model(input_shape):
    input_layer = Input(shape=input_shape)
    conv_layer = Conv1D(filters=64, kernel_size=3, activation='relu')(input_layer)
    flatten_layer = Flatten()(conv_layer)
    output_layer = Dense(1, activation='sigmoid')(flatten_layer)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return model

def create_lstm_model(input_shape):
    input_layer = Input(shape=input_shape)
    lstm_layer = LSTM(64)(input_layer)
    output_layer = Dense(1, activation='sigmoid')(lstm_layer)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return model

def create_dffnn_model(input_dim):
    input_layer = Input(shape=(input_dim,))
    dense_layer = Dense(64, activation='relu')(input_layer)
    output_layer = Dense(1, activation='sigmoid')(dense_layer)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
    return model

input_dim = X_train_scaled.shape[1]
cnn_model = create_cnn_model((input_dim, 1))
lstm_model = create_lstm_model((input_dim, 1))
dffnn_model = create_dffnn_model(input_dim)

epochs = 5
batch_size = 32

print("Training CNN Model...")
cnn_model.fit(X_train_scaled_reshaped, y_train, epochs=epochs, batch_size=batch_size, verbose=1)

print("\nTraining LSTM Model...")
lstm_model.fit(X_train_scaled_reshaped, y_train, epochs=epochs, batch_size=batch_size, verbose=1)

# Evaluate models before SHAP computations
print("\nEvaluating models before SHAP computations...")
cnn_loss, cnn_accuracy = cnn_model.evaluate(X_test_scaled_reshaped, y_test, verbose=0)
lstm_loss, lstm_accuracy = lstm_model.evaluate(X_test_scaled_reshaped, y_test, verbose=0)
dffnn_loss, dffnn_accuracy = dffnn_model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"CNN Test Accuracy: {cnn_accuracy:.4f}")
print(f"LSTM Test Accuracy: {lstm_accuracy:.4f}")
print(f"D-FFNN Test Accuracy: {dffnn_accuracy:.4f}")

# ----------------------
# 3. COMPUTE SHAP VALUES AND FEATURE IMPORTANCE
# ----------------------

def compute_shap_values():
    # Initialize SHAP
    shap.initjs()

    # Use a subset of your training data as background data
    background_size = min(100, X_train_scaled.shape[0])
    background_dffnn = X_train_scaled[np.random.choice(X_train_scaled.shape[0], background_size, replace=False)]
    background_cnn_lstm = X_train_scaled_reshaped[np.random.choice(X_train_scaled_reshaped.shape[0], background_size, replace=False)]

    # Number of test samples to explain
    num_test_samples = min(100, X_test_scaled.shape[0])

    # Prepare list to store SHAP values
    shap_values_list = []

    # ----------------------
    # SHAP for D-FFNN Model
    # ----------------------
    print("\nComputing SHAP values for D-FFNN Model...")
    explainer_dffnn = shap.GradientExplainer(dffnn_model, background_dffnn)
    shap_values_dffnn = explainer_dffnn.shap_values(X_test_scaled[:num_test_samples])
    shap_values_dffnn = np.array(shap_values_dffnn).squeeze()  # Shape: (num_samples, num_features)
    shap_values_list.append(shap_values_dffnn)

    # ----------------------
    # SHAP for CNN Model
    # ----------------------
    print("\nComputing SHAP values for CNN Model...")
    explainer_cnn = shap.GradientExplainer(cnn_model, background_cnn_lstm)
    shap_values_cnn = explainer_cnn.shap_values(X_test_scaled_reshaped[:num_test_samples])
    shap_values_cnn = np.array(shap_values_cnn).squeeze()

    # Handle SHAP values shape for CNN
    if len(shap_values_cnn.shape) == 3:
        # Aggregate over the convolutional steps
        shap_values_cnn = np.mean(shap_values_cnn, axis=1)
    shap_values_list.append(shap_values_cnn)

    # ----------------------
    # SHAP for LSTM Model
    # ----------------------
    print("\nComputing SHAP values for LSTM Model...")
    explainer_lstm = shap.GradientExplainer(lstm_model, background_cnn_lstm)
    shap_values_lstm = explainer_lstm.shap_values(X_test_scaled_reshaped[:num_test_samples])
    shap_values_lstm = np.array(shap_values_lstm).squeeze()
    shap_values_list.append(shap_values_lstm)

    return shap_values_list, num_test_samples

# Compute SHAP values
shap_values_list, num_test_samples = compute_shap_values()

# Clear the Keras backend session
K.clear_session()

# ----------------------
# 4. COMPUTE AVERAGE SHAP VALUES
# ----------------------

# Convert list of SHAP values to NumPy array
shap_values_array = np.array(shap_values_list)  # Shape: (3, num_samples, num_features)

# Compute the average SHAP values across models
average_shap_values = np.mean(shap_values_array, axis=0)  # Shape: (num_samples, num_features)
```

```
# Convert list of SHAP values to NumPy array
shap_values_array = np.array(shap_values_list)  # Shape: (3, num_samples, num_features)

# Compute the average SHAP values across models
average_shap_values = np.mean(shap_values_array, axis=0)  # Shape: (num_samples, num_features)

# ----------------------
# 5. VISUALIZE THE AVERAGE FEATURE IMPORTANCE AND SHAP BEE SWARM PLOT
# ----------------------

# Plot Average Importance
average_importance = np.mean(np.abs(average_shap_values), axis=0)
importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Average Importance': average_importance
})

# Sort features by average importance
importances_df = importances_df.sort_values(by='Average Importance', ascending=False).reset_index(drop=True)

plt.figure(figsize=(20, 14))
plt.bar(importances_df['Feature'], importances_df['Average Importance'], color='SkyBlue')
plt.xticks(rotation=90)
plt.xlabel('Features')
plt.ylabel('Average Mean Absolute SHAP Value')
plt.title('Average Feature Importance Across Models (Using Mean)')
plt.tight_layout()
plt.show()

# SHAP Bee Swarm Plot for Average SHAP Values
print("\nCreating SHAP Bee Swarm Plot for Average SHAP Values...")
plt.figure(figsize=(20, 16))
shap.summary_plot(average_shap_values, X_test_scaled[:num_test_samples], feature_names=feature_names)
```