```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from utils import load_data, load_topology
import matplotlib.pyplot as plt
import networkx as nx
from sklearn.tree import plot_tree
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier

def setup_seed(seed):
    np.random.seed(seed)
    import random
    random.seed(seed)

def plot_network_with_labels(adj, non_zero_index):
    # Remove self-loops from the adjacency matrix
    np.fill_diagonal(adj, 0)

    # Create a NetworkX graph from the adjacency matrix
    g = nx.from_numpy_array(adj)

    # Adjust layout to spread nodes
    pos = nx.spring_layout(g, seed=42, k=0.25)  # Smaller k value spreads nodes
more

    # Map feature IDs to nodes and links
    feature_mapping = {}
    feature_id = 1

    # Node feature IDs (1-150)
    for i in range(adj.shape[0]):
        feature_mapping[f"{i + 1}"] = feature_id  # Simplified label for nodes
        feature_id += 1

    # Link feature IDs (151-353)
    for i, j in zip(non_zero_index[0], non_zero_index[1]):
        feature_mapping[f"Link {i + 1}-{j + 1}"] = feature_id
        feature_id += 1

    # Plot the graph
    plt.figure(figsize=(16, 16))
    nx.draw_networkx_edges(g, pos, alpha=0.5)
    nx.draw_networkx_nodes(g, pos, node_size=150, node_color='blue')  # Smaller
node size

    # Add simplified labels for nodes (just their IDs)
    node_labels = {i: f"{i + 1}" for i in range(adj.shape[0])}
    nx.draw_networkx_labels(g, pos, labels=node_labels, font_color='red',
font_size=8)

    # Add labels for edges (links)
    edge_labels = {edge: f"{feature_mapping[f'Link {edge[0] + 1}-{edge[1] + 1}']}"
for edge in g.edges}
    nx.draw_networkx_edge_labels(g, pos, edge_labels=edge_labels,
```

```
font_color='green', font_size=8)

    plt.title("Power Network with Node and Link IDs")
    plt.axis("off")
    plt.savefig("power_network.png")
    # plt.show()

if __name__ == '__main__':
    # Set random seed for reproducibility
    setup_seed(2022)

    # Load power information including before/after matrices and labels
    split_list = [100, 1000]
    before_array, after_array, y = load_data(split_list)

    # Load the adjacency matrix of the 150-node power system
    adj = load_topology()

    # Add self-loops
    adj = adj + np.eye(adj.shape[0])

    # Obtain non-zero indices from the adjacency matrix (upper triangular to avoid
redundancy)
    triangle_adj = np.triu(adj)
    non_zero_index = np.nonzero(triangle_adj)

    # Plot the network
    print("Plotting the power network with node and link IDs...")
    plot_network_with_labels(adj, non_zero_index)

    # Generate the difference matrix
    X = after_array - before_array

    # Filter non-zero elements based on the adjacency matrix
    print("Filtering non-zero elements based on the adjacency matrix...")
    X_filtered = X[:, non_zero_index[0], non_zero_index[1]]

    # Reshape filtered data into a 2D feature array for the decision tree (samples
x features)
    # Each row corresponds to a scenario, and columns are the non-zero features
    print("Reshaping filtered data into a 2D feature array...")
    X_features = X_filtered.reshape(X_filtered.shape[0], -1)

    print("X_features shape:", X_features.shape)

    # Split data into training and testing sets
    idx = np.arange(X_features.shape[0])
    idx_train, idx_test, y_train, y_test = train_test_split(idx, y, test_size=0.3,
stratify=y)

    # Use the filtered features for training and testing
    X_train = X_features[idx_train]
    X_test = X_features[idx_test]

    print()

    # # Decision Tree
================================================================================
    # print("Decision Tree Classifier:")
```

```
    # # Initialize and train the decision tree classifier
    # clf = DecisionTreeClassifier(max_depth=10, class_weight='balanced',
random_state=2022)
    # clf.fit(X_train, y_train)

    # # Evaluate the classifier on training and testing sets
    # y_train_pred = clf.predict(X_train)
    # y_test_pred = clf.predict(X_test)

    # train_acc = accuracy_score(y_train, y_train_pred)
    # test_acc = accuracy_score(y_test, y_test_pred)

    # print(f"Training Accuracy: {train_acc}")
    # print(f"Testing Accuracy: {test_acc}")

    # # Print precision, recall, and F1-score
    # print("Classification Report (Training):")
    # print(classification_report(y_train, y_train_pred))

    # print("Classification Report (Testing):")
    # print(classification_report(y_test, y_test_pred))

    # print("Confusion Matrix (Training):")
    # conf_matrix = confusion_matrix(y_train, y_train_pred)
    # print("Confusion Matrix:\n", conf_matrix)

    # print("Confusion Matrix (Testing):")
    # conf_matrix = confusion_matrix(y_test, y_test_pred)
    # print("Confusion Matrix:\n", conf_matrix)

    # # (Optional) Feature importance
    # feature_importances = clf.feature_importances_
    # # print("Feature Importances (top 10):", np.argsort(feature_importances)[-
10:])
    # sorted_indices = np.argsort(feature_importances)[::-1]
    # print("Feature Importances (sorted by importance):", sorted_indices[:10])

    # # Plot the decision tree
    # plt.figure(figsize=(20, 10))  # Adjust figure size as needed
    # plot_tree(
    #     clf,
    #     feature_names=[f"Feature {i}" for i in range(X_train.shape[1])],
    #     # class_names=[f"Class {i}" for i in np.unique(y_train)],
    #     filled=True,
    #     rounded=True,
    #     proportion=False,
    #     fontsize=8,
    # )
    # plt.title("Decision Tree Visualization")
    # plt.savefig("decision_tree.png")

    # print()

    # # SVM
================================================================================
=========
    # print("SVM Classifier:")
    # # Standardize features for SVM
    # scaler = StandardScaler()
```

```
    # X_train_scaled = scaler.fit_transform(X_train)
    # X_test_scaled = scaler.transform(X_test)

    # # Initialize and train the SVM classifier
    # svm_clf = SVC(kernel='rbf', C=1.0, gamma='scale', class_weight='balanced',
random_state=2022)
    # svm_clf.fit(X_train_scaled, y_train)

    # # Evaluate the classifier on training and testing sets
    # y_train_pred = svm_clf.predict(X_train_scaled)
    # y_test_pred = svm_clf.predict(X_test_scaled)

    # train_acc = accuracy_score(y_train, y_train_pred)
    # test_acc = accuracy_score(y_test, y_test_pred)

    # print(f"Training Accuracy: {train_acc}")
    # print(f"Testing Accuracy: {test_acc}")

    # # Print precision, recall, and F1-score
    # print("Classification Report (Training):")
    # print(classification_report(y_train, y_train_pred))

    # print("Classification Report (Testing):")
    # print(classification_report(y_test, y_test_pred))

    # print("Confusion Matrix (Training):")
    # conf_matrix = confusion_matrix(y_train, y_train_pred)
    # print("Confusion Matrix:\n", conf_matrix)

    # print("Confusion Matrix (Testing):")
    # conf_matrix = confusion_matrix(y_test, y_test_pred)
    # print("Confusion Matrix:\n", conf_matrix)

    # KNN Classifier
================================================================================
    print("KNN Classifier:")

    # Standardize features for KNN
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Initialize and train the KNN classifier
    # Set the number of neighbors (k) to 5, which is a common choice; you can
experiment with different values
    knn_clf = KNeighborsClassifier(n_neighbors=5, weights='distance',
metric='minkowski', p=2)

    # Fit the classifier to the training data
    knn_clf.fit(X_train_scaled, y_train)

    # Evaluate the classifier on training and testing sets
    y_train_pred = knn_clf.predict(X_train_scaled)
    y_test_pred = knn_clf.predict(X_test_scaled)

    # Calculate training and testing accuracy
    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)
```

```
print(f"Training Accuracy: {train_acc}")
print(f"Testing Accuracy: {test_acc}")

# Print precision, recall, and F1-score
print("Classification Report (Training):")
print(classification_report(y_train, y_train_pred))

print("Classification Report (Testing):")
print(classification_report(y_test, y_test_pred))

print("Confusion Matrix (Training):")
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Confusion Matrix:\n", conf_matrix)

print("Confusion Matrix (Testing):")
conf_matrix = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:\n", conf_matrix)
```