%% Fault Detection in Three-Phase Multilevel Inverters Using Machine Learning

% This code implements various machine learning models to detect and classify

% faults in multilevel inverters for smart grid applications.

clc;

clear;

close all;

%% 1. Data Generation and Feature Extraction

% Simulate normal and fault conditions in a 3-level NPC inverter

% Parameters

fs = 20e3; % Sampling frequency (Hz)

T = 1/fs; % Sampling period

t = 0:T:0.2-T; % Time vector (0.1 second)

f = 50; % Fundamental frequency (Hz)

Vdc = 400; % DC link voltage (V)

m = 0.85; % Modulation index

% Generate reference signals

Vref = m \* sin(2\*pi\*f\*t); % Reference sine wave

triang = sawtooth(2\*pi\*20\*t, 0.5); % Carrier wave (triangular)

% Initialize output voltage

Vout = zeros(size(t));

% Normal operation PWM generation

for i = 1:length(t)

if Vref(i) > triang(i)

Vout(i) = Vdc/2;

elseif Vref(i) < -triang(i)

Vout(i) = -Vdc/2;

else

Vout(i) = 0;

end

end

% Add noise to simulate real conditions

Vout = Vout + 0.02\*Vdc\*randn(size(Vout));

% Create fault conditions

% Fault 1: Lower MOSFETs isolated (Scenario I)

Vout\_fault1 = Vout;

fault\_start = floor(length(t)/3);

fault\_end = floor(2\*length(t)/3);

Vout\_fault1(fault\_start:fault\_end) = Vout\_fault1(fault\_start:fault\_end) + 0.5\*Vdc;

% Fault 2: Lower MOSFETs short-circuited (Scenario II)

Vout\_fault2 = Vout;

Vout\_fault2(fault\_start:fault\_end) = Vout\_fault2(fault\_start:fault\_end) - 0.7\*Vdc;

% Fault 3: Upper MOSFETs isolated (Scenario III)

Vout\_fault3 = Vout;

Vout\_fault3(fault\_start:fault\_end) = Vout\_fault3(fault\_start:fault\_end) + 0.8\*Vdc;

% Fault 4: Upper MOSFETs short-circuited (Scenario IV)

Vout\_fault4 = Vout;

Vout\_fault4(fault\_start:fault\_end) = Vout\_fault4(fault\_start:fault\_end) - 0.6\*Vdc;

%% Feature Extraction using FFT and THD Calculation

% Function to calculate THD and extract harmonic features

function [thd, features] = extractFeatures(signal, fs, f)

N = length(signal);

fft\_signal = abs(fft(signal)/N);

fft\_signal = fft\_signal(1:N/2+1);

fft\_signal(2:end-1) = 2\*fft\_signal(2:end-1);

f\_axis = fs\*(0:(N/2))/N;

% Find fundamental frequency component

[~, fund\_idx] = min(abs(f\_axis - f));

fund\_amp = fft\_signal(fund\_idx);

% Calculate THD

harmonic\_bands = (2:40)\*f; % Up to 40th harmonic

harmonic\_power = 0;

for h = harmonic\_bands

[~, h\_idx] = min(abs(f\_axis - h));

if h\_idx <= length(fft\_signal)

harmonic\_power = harmonic\_power + fft\_signal(h\_idx)^2;

end

end

thd = sqrt(harmonic\_power)/fund\_amp \* 100;

% Extract harmonic features (1st to 5th harmonics)

features = zeros(1,5);

for h = 1:5

[~, h\_idx] = min(abs(f\_axis - h\*f));

if h\_idx <= length(fft\_signal)

features(h) = fft\_signal(h\_idx)/fund\_amp;

end

end

% Add statistical features

features = [features, mean(signal), std(signal), skewness(signal), kurtosis(signal)];

end

% Extract features for all conditions

[thd\_normal, features\_normal] = extractFeatures(Vout, fs, f);

[thd\_fault1, features\_fault1] = extractFeatures(Vout\_fault1, fs, f);

[thd\_fault2, features\_fault2] = extractFeatures(Vout\_fault2, fs, f);

[thd\_fault3, features\_fault3] = extractFeatures(Vout\_fault3, fs, f);

[thd\_fault4, features\_fault4] = extractFeatures(Vout\_fault4, fs, f);

% Display THD values

fprintf('THD Values:\n');

fprintf('Normal operation: %.2f%%\n', thd\_normal);

fprintf('Lower MOSFETs isolated: %.2f%%\n', thd\_fault1);

fprintf('Lower MOSFETs short-circuited: %.2f%%\n', thd\_fault2);

fprintf('Upper MOSFETs isolated: %.2f%%\n', thd\_fault3);

fprintf('Upper MOSFETs short-circuited: %.2f%%\n', thd\_fault4);

%% 2. Create Dataset for Machine Learning

% Generate multiple samples for each condition with slight variations

num\_samples = 100;

features = zeros(num\_samples\*5, 9); % 5 conditions, 9 features each

labels = zeros(num\_samples\*5, 1);

for i = 1:num\_samples

% Normal operation with random variations

noise\_level = 0.01 + 0.01\*rand();

V\_sample = Vout + noise\_level\*Vdc\*randn(size(Vout));

[~, features((i-1)\*5+1,:)] = extractFeatures(V\_sample, fs, f);

labels((i-1)\*5+1) = 0;

% Fault 1 with random variations

V\_sample = Vout\_fault1 + noise\_level\*Vdc\*randn(size(Vout));

[~, features((i-1)\*5+2,:)] = extractFeatures(V\_sample, fs, f);

labels((i-1)\*5+2) = 1;

% Fault 2 with random variations

V\_sample = Vout\_fault2 + noise\_level\*Vdc\*randn(size(Vout));

[~, features((i-1)\*5+3,:)] = extractFeatures(V\_sample, fs, f);

labels((i-1)\*5+3) = 2;

% Fault 3 with random variations

V\_sample = Vout\_fault3 + noise\_level\*Vdc\*randn(size(Vout));

[~, features((i-1)\*5+4,:)] = extractFeatures(V\_sample, fs, f);

labels((i-1)\*5+4) = 3;

% Fault 4 with random variations

V\_sample = Vout\_fault4 + noise\_level\*Vdc\*randn(size(Vout));

[~, features((i-1)\*5+5,:)] = extractFeatures(V\_sample, fs, f);

labels((i-1)\*5+5) = 4;

end

% Split into training and testing sets (70/30 split)

rng(42); % For reproducibility

cv = cvpartition(labels, 'HoldOut', 0.3);

X\_train = features(cv.training,:);

y\_train = labels(cv.training,:);

X\_test = features(cv.test,:);

y\_test = labels(cv.test,:);

%% 3. Train and Evaluate Multiple Machine Learning Models

% Model 1: Decision Tree

disp('Training Decision Tree...');

tree = fitctree(X\_train, y\_train, 'OptimizeHyperparameters', 'auto');

y\_pred\_tree = predict(tree, X\_test);

acc\_tree = sum(y\_pred\_tree == y\_test)/length(y\_test) \* 100;

fprintf('Decision Tree Accuracy: %.2f%%\n', acc\_tree);

% Model 2: Support Vector Machine (SVM)

disp('Training SVM...');

svm = fitcecoc(X\_train, y\_train, 'OptimizeHyperparameters', 'auto');

y\_pred\_svm = predict(svm, X\_test);

acc\_svm = sum(y\_pred\_svm == y\_test)/length(y\_test) \* 100;

fprintf('SVM Accuracy: %.2f%%\n', acc\_svm);

% Model 3: Neural Network

disp('Training Neural Network...');

net = patternnet(10); % Single hidden layer with 10 neurons

net.trainParam.showWindow = false; % Suppress training GUI

net = train(net, X\_train', dummyvar(y\_train+1)');

y\_pred\_nn = net(X\_test');

[~, y\_pred\_nn] = max(y\_pred\_nn);

y\_pred\_nn = y\_pred\_nn' - 1;

acc\_nn = sum(y\_pred\_nn == y\_test)/length(y\_test) \* 100;

fprintf('Neural Network Accuracy: %.2f%%\n', acc\_nn);

% Model 4: Random Forest

disp('Training Random Forest...');

rf = TreeBagger(50, X\_train, y\_train, 'Method', 'classification');

y\_pred\_rf = str2double(predict(rf, X\_test));

acc\_rf = sum(y\_pred\_rf == y\_test)/length(y\_test) \* 100;

fprintf('Random Forest Accuracy: %.2f%%\n', acc\_rf);

% Model 5: k-Nearest Neighbors (k-NN)

disp('Training k-NN...');

knn = fitcknn(X\_train, y\_train, 'OptimizeHyperparameters', 'auto');

y\_pred\_knn = predict(knn, X\_test);

acc\_knn = sum(y\_pred\_knn == y\_test)/length(y\_test) \* 100;

fprintf('k-NN Accuracy: %.2f%%\n', acc\_knn);

%% 4. Compare Model Performance and Select the Best One

model\_names = {'Decision Tree', 'SVM', 'Neural Network', 'Random Forest', 'k-NN'};

accuracies = [acc\_tree, acc\_svm, acc\_nn, acc\_rf, acc\_knn];

figure;

bar(accuracies);

title('Model Comparison - Classification Accuracy');

set(gca, 'XTickLabel', model\_names);

ylabel('Accuracy (%)');

grid on;

% Find the best model

[best\_acc, best\_idx] = max(accuracies);

fprintf('\nBest model: %s with %.2f%% accuracy\n', model\_names{best\_idx}, best\_acc);

%% 5. Confusion Matrix for the Best Model

switch best\_idx

case 1

best\_model = tree;

y\_pred = y\_pred\_tree;

case 2

best\_model = svm;

y\_pred = y\_pred\_svm;

case 3

y\_pred = y\_pred\_nn;

case 4

best\_model = rf;

y\_pred = y\_pred\_rf;

case 5

best\_model = knn;

y\_pred = y\_pred\_knn;

end

figure;

confusionchart(y\_test, y\_pred);

title(['Confusion Matrix for ', model\_names{best\_idx}]);

xlabel('Predicted Class');

ylabel('True Class');

%% 6. Feature Importance Analysis (for tree-based models)

if ismember(best\_idx, [1,4]) % Decision Tree or Random Forest

figure;

if best\_idx == 1

imp = predictorImportance(tree);

else

imp = rf.OOBPermutedPredictorDeltaError;

end

bar(imp);

title('Feature Importance');

xlabel('Features');

ylabel('Importance Score');

xticklabels({'1st Harm', '2nd Harm', '3rd Harm', '4th Harm', '5th Harm', ...

'Mean', 'Std Dev', 'Skewness', 'Kurtosis'});

xtickangle(45);

grid on;

end

%% 7. Visualize Fault Signatures

figure;

subplot(3,2,1);

plot(t, Vout);

title('Normal Operation');

xlabel('Time (s)');

ylabel('Voltage (V)');

grid on;

subplot(3,2,2);

plot(t, Vout\_fault1);

title('Lower MOSFETs Isolated');

xlabel('Time (s)');

ylabel('Voltage (V)');

grid on;

subplot(3,2,3);

plot(t, Vout\_fault2);

title('Lower MOSFETs Short-Circuited');

xlabel('Time (s)');

ylabel('Voltage (V)');

grid on;

subplot(3,2,4);

plot(t, Vout\_fault3);

title('Upper MOSFETs Isolated');

xlabel('Time (s)');

ylabel('Voltage (V)');

grid on;

subplot(3,2,5);

plot(t, Vout\_fault4);

title('Upper MOSFETs Short-Circuited');

xlabel('Time (s)');

ylabel('Voltage (V)');

grid on;

%% 8. FFT Analysis Visualization

[~, fft\_normal] = extractFeatures(Vout, fs, f);

[~, fft\_fault1] = extractFeatures(Vout\_fault1, fs, f);

[~, fft\_fault2] = extractFeatures(Vout\_fault2, fs, f);

[~, fft\_fault3] = extractFeatures(Vout\_fault3, fs, f);

[~, fft\_fault4] = extractFeatures(Vout\_fault4, fs, f);

figure;

subplot(2,1,1);

bar([fft\_normal(1:5); fft\_fault1(1:5); fft\_fault2(1:5); fft\_fault3(1:5); fft\_fault4(1:5)]);

title('Harmonic Components (1st-5th)');

legend('1st', '2nd', '3rd', '4th', '5th');

set(gca, 'XTickLabel', {'Normal', 'Fault1', 'Fault2', 'Fault3', 'Fault4'});

ylabel('Normalized Amplitude');

grid on;

subplot(2,1,2);

bar([thd\_normal, thd\_fault1, thd\_fault2, thd\_fault3, thd\_fault4]);

title('Total Harmonic Distortion (THD)');

set(gca, 'XTickLabel', {'Normal', 'Fault1', 'Fault2', 'Fault3', 'Fault4'});

ylabel('THD (%)');

grid on;

%