```
degradationData = helperLoadData('train.txt');
degradationData(1:5)

head(degradationData{1})
rng('default')
numEnsemble = length(degradationData);
numFold = 5;
cv = cvpartition(numEnsemble, 'KFold', numFold);
trainData = degradationData(training(cv, 1));
validationData = degradationData(test(cv, 1));

varNames = string(degradationData{1}.Properties.VariableNames);
timeVariable = varNames{2};
conditionVariables = varNames(3:5);
dataVariables = varNames(6:26);

nsample = 10;
figure
helperPlotEnsemble(trainData, timeVariable, ...
    [conditionVariables(1:2) dataVariables(1:2)], nsample)

trainDataUnwrap = vertcat(trainData{:});
opConditionUnwrap = trainDataUnwrap(:, cellstr(conditionVariables));

figure
helperPlotClusters(opConditionUnwrap)

opts = statset('Display', 'final');
[clusterIndex, centers] = kmeans(table2array(opConditionUnwrap), 6, ...
    'Distance', 'sqeuclidean', 'Replicates', 5, 'Options', opts);

figure
helperPlotClusters(opConditionUnwrap, clusterIndex, centers)

centerstats = struct('Mean', table(), 'SD', table());
for v = dataVariables
    centerstats.Mean.(char(v)) = splitapply(@mean, trainDataUnwrap.(char(v)), clusterIndex);
    centerstats.SD.(char(v)) = splitapply(@std, trainDataUnwrap.(char(v)), clusterIndex);
end
centerstats.Mean

centerstats.SD

trainDataNormalized = cellfun(@(data) regimeNormalization(data, centers, centerstats), ...
    trainData, 'UniformOutput', false);

figure
helperPlotEnsemble(trainDataNormalized, timeVariable, dataVariables(1:4), nsample)

numSensors = length(dataVariables);
signalSlope = zeros(numSensors, 1);
warn = warning('off');
for ct = 1:numSensors
```

```
    tmp = cellfun(@(tbl) tbl(:, cellstr(dataVariables(ct))), trainDataNormalized, 'UniformOutput', false);
    mdl = linearDegradationModel(); %create model
    fit(mdl, tmp); % train mode
    signalSlope(ct) = mdl.Theta;
end
warning(warn);

[~, idx] = sort(abs(signalSlope), 'descend');
sensorTrended = sort(idx(1:8))

figure
helperPlotEnsemble(trainDataNormalized, timeVariable, dataVariables(sensorTrended(3:6)), nsample)

%%
numHidden = ([30,20,20]);
maxEpochs = 10000;
mseTarget =  1e-5;
eta = 0.03;

net = patternnet(numHidden);
net = configure(net,inputs,targets);
net.inputWeights(:,:).initFcn = 'initnw';
net.layerWeights(:,:).initFcn = 'initnw';
net.biases(:).initFcn = 'initnw';

net.trainFcn = 'train'
net.trainParam.epochs = maxEpochs;
net.trainParam.goal = mseTarget;
net.trainParam.showWindow;
net.trainParam.lr = eta;
net.trainParam.min_grad = 10^-5;
net.performFcn = 'mse';
net.divideParam.trainRatio = 5/8;
net.divideParam.valRatio = 1/8;
net.divideParam.testRatio = 2/8;

[net,tr] = train(net,inputs,targets);

out = net(inputs);
outputs = hardlim(out-thresh);
errora = gaubtract(targets,outputs);

view(net)

figure, plotperform(tr)
figure, plotconfusion(targets,outputs)

%%
for j=1:numel(trainDataNormalized)
    data = trainDataNormalized{j};
    rul = max(data.time)-data.time;
    data.health_condition = rul / max(rul);
    trainDataNormalized{j} = data;
end
```

```
figure
helperPlotEnsemble(trainDataNormalized, timeVariable, "health_condition", nsample)

trainDataNormalizedUnwrap = vertcat(trainDataNormalized{:});

sensorToFuse = dataVariables(sensorTrended);
X = trainDataNormalizedUnwrap{:, cellstr(sensorToFuse)};
y = trainDataNormalizedUnwrap.health_condition;
regModel = fitlm(X,y);
bias = regModel.Coefficients.Estimate(1)

weights = regModel.Coefficients.Estimate(2:end)

trainDataFused = cellfun(@(data) degradationSensorFusion(data, sensorToFuse, weights), trainDataNormalized, ...
    'UniformOutput', false);

figure
helperPlotEnsemble(trainDataFused, [], 1, nsample)
xlabel('Time')
ylabel('Health Indicator')
title('Training Data')

validataionDataNormalized = cellfun(@(data) regimeNormalization(data, centers, centerstats), ...
    validationData, 'UniformOutput', false);
validationDataFused = cellfun(@(data) degradationSensorFusion(data, sensorToFuse, weights), ...
    validationDataNoramlized, 'UniformOutput', false);

figure
helperPlotEnsemble(validationDataFused, [], 1, nsample)
xlabel('Time')
ylabel('Health Indicator')
title('Validation Data')

mdl = residualSimilarityModel(...
    'Method', 'poly2',...
    'Distance', 'absolute',...
    'NumNearestNeighbors', 50,...
    'Standardize', 1);

fit(mdl, trainDataFused);

breakpoint = [0.5, 0.7, 0.9];
validationDataTmp = validationDataFused{3}; % use one validation data for illustration

bpidx = 1;
validationDataTmp50 = validationDataTmp(1:ceil(end*breakpoint(bpidx)),:);
trueRUL = length(validationDataTmp) - length(validationDataTmp50);
[estRUL, ciRUL, pdfRUL] = predictRUL(mdl, validationEDataTmp50);

figure
compare(mdl, validationDataTmp50);

figure
```

```
helperPlotRULDistribution(trueRUL, estRUL, pdfRUL, ciRUL)

bpidx = 2;
validationDataTmp70 = validationDataTmp(1:ceil(end*breakpoint(bpidx)), :);
trueRUL = length(validationDataTmp) - length(validationDataTmp70);
[estRUL,ciRUL,pdfRUL] = predictRUL(mdl, validationDataTmp70);

figure
compare(mdl, validationDataTmp70);

figure
helperPlotRULDistribution(trueRUL, estRUL, pdfRUL, ciRUL)

bpidx = 3;
validationDataTmp90 = validationDataTmp(1:ceil(end*breakpoint(bpidx)), :);
trueRUL = length(validationDataTmp) - length(validationDataTmp90);
[estRUL,ciRUL,pdfRUL] = predictRUL(mdl, validationDataTmp90);

figure
compare(mdl, validationDataTmp90);

figure
helperPlotRULDistribution(trueRUL, estRUL, pdfRUL, ciRUL);

numValidation = length(validationDataFused);
numBreakpoint = length(breakpoint);
error = zeros(numValidation, numBreakpoint);

for dataIdx = 1:numValidation
    tmpData = validationDataFused{dataIdx};
    for bpidx = 1:numBreakpoint
        tmpDataTest = tmpData(1:ceil(end*breakpoint(bpidx)), :);
        trueRUL = length(tmpData) - length(tmpDataTest);
        [estRUL, ~, ~] = predictRUL(mdl, tmpDataTest);
        error(dataIdx, bpidx) = estRUL - trueRUL;
    end
end

[pdf50, x50] = ksdensity(error(:, 1));
[pdf70, x70] = ksdensity(error(:, 2));
[pdf90, x90] = ksdensity(error(:, 3));

figure
ax(1) = subplot(3,1,1);
hold on
histogram(error(:, 1), 'BinWidth', 5, 'Normalization', 'pdf')
plot(x50, pdf50)
hold off
xlabel('Prediction Error')
title('RUL Prediction Error using first 50% of each validation ensemble member')

ax(2) = subplot(3,1,2);
hold on
histogram(error(:, 2), 'BinWidth', 5, 'Normalization', 'pdf')
```

```
plot(x70, pdf70)
hold off
xlabel('Prediction Error')
title('RUL Prediction Error using first 70% of each validation ensemble member')

ax(3) = subplot(3,1,3);
hold on
histogram(error(:, 3), 'BinWidth', 5, 'Normalization', 'pdf')
plot(x90, pdf90)
hold off
xlabel('Prediction Error')
title('RUL Prediction Error using first 90% of each validation ensemble member')
linkaxes(ax)

figure
boxplot(error, 'Labels', {'50%', '70%', '90%'})
ylabel('Prediction Error')
title('Prediction error using different percentages of each validation ensemble member')

errorMean = mean(error)

errorMedian = median(error)

errorSD = std(error)

figure
errorbar([50 70 90], errorMean, errorSD, '-o', 'MarkerEdgeColor','r')
xlim([40, 100])
xlabel('Percentage of validation data used for RUL prediction')
ylabel('Prediction Error')
legend('Mean Prediction Error with 1 Standard Deviation Error Bar', 'Location', 'south')
```