```
clc
clear all
```

Load the saved preprocessed data. Two class corresponds to "REM" vs "Wake" EEG signal

```
Training = load('C:\courses\ML\project\codes\TrainNetworkProject2\dataset_processed\TwoclassTra
Training = Training.Balancedsleep2ClasstrainUSCell;
TrainingLabels = load('C:\courses\ML\project\codes\TrainNetworkProject2\dataset_processed\Twocl
TrainingLabels = TrainingLabels.TwoClassTrain_US_Labels';
Validation= load('C:\courses\ML\project\codes\TrainNetworkProject2\dataset_processed\TwoclassVa
Validation = Validation.Balancedsleep2ClassvalidationUSCell;
ValidationLabels = load('C:\courses\ML\project\codes\TrainNetworkProject2\dataset_processed\Two
ValidationLabels = ValidationLabels.TwoClassValidation_US_Labels';
Testing = load('C:\courses\ML\project\codes\TrainNetworkProject2\dataset_processed\TwoclassTest
 Testing = Testing.Balancedsleep2ClasstestUSCell;
TestingLabels = load('C:\courses\ML\project\codes\TrainNetworkProject2\dataset_processed\Twocla
TestingLabels = TestingLabels.TwoClassTest_US_Labels';
```

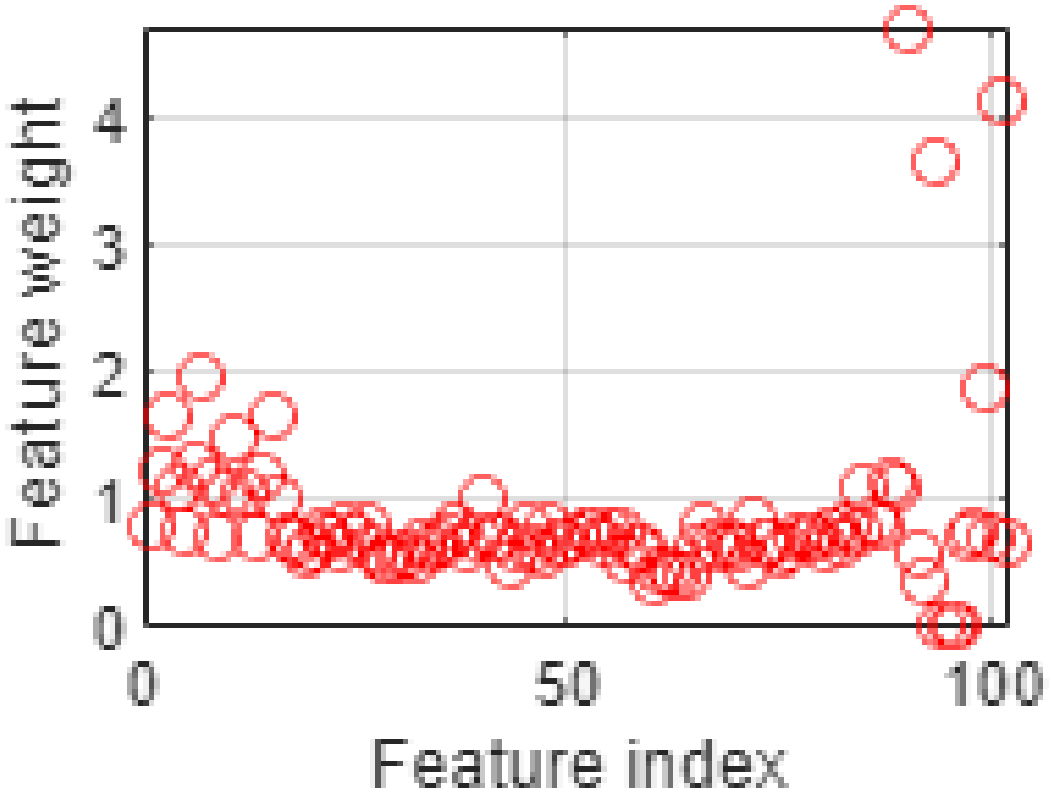
Extract features: Here we consider variety of features used for Non stationary signal analysis such an EEG signal.The features are 1) Autoregressive model coefficients (of order 4) estimated using Burg method, Shannon entropy for the maximum overlap discrete wavelet packet transform, The singularity spectrum (Multifractal wavelet leader estimate of the second cumulant of the scaling exponents and the range of holder exponents, Wavelet variance, Hijorth parameters, mean instantaneous frequency and standard features such as mean, variance, kurtosis of the signal) . The time window chosen is 250.

```
timeWindow = 250;
ARorder = 4;
MODWPTlevel = 4;
[trainFeatures,ValidationFeatures, testFeatures, featureindices] = helperExtractFeatures(Traini
```

Computing feature weights by fitting a Neighbourhood component analysis (NCA) model

```
train_reduced = fscnca(trainFeatures,TrainingLabels);

figure()
plot(train_reduced.FeatureWeights,'ro')
grid on
xlabel('Feature index')
ylabel('Feature weight')
```



Feature selection using the tolerance level "tol". Currently we kept it as 0.1 in the interest of time. But it can also be set as a hyperparameter.

```
tol    = 0.1;
selidx = find(train_reduced.FeatureWeights > tol*max(1,max(train_reduced.FeatureWeights)));
trainFeatures1 = trainFeatures(:,selidx);
ValidationFeatures1 = ValidationFeatures(:,selidx);
testFeatures1 = testFeatures(:,selidx);


features = [trainFeatures1; ValidationFeatures1];
labels = [TrainingLabels;ValidationLabels];
```

Fit an SVM model and perform hyperparameter tuning on parameters such as Kernel (polynomial, guassian, linear), polynomial order, kernel scale etc.

```
rng(1)
model = fitcecoc(...
    features,...
    labels,...
    'Learners','svm',...
    'Coding','onevsone',...
    'ClassNames',{'REM','Wake'},'OptimizeHyperparameters','all',...
    'HyperparameterOptimizationOptions',struct('MaxTime',4*60*60,'AcquisitionFunctionName',...
    'expected-improvement-plus'));
```

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Coding	BoxConstraint	KernelScale
1	Best	0.15994	20.522	0.15994	0.15994	onevsone	0.0049535	-
2	Accept	0.37086	1194.6	0.15994	0.22232	onevsall	0.017242	-
3	Best	0.10801	50.294	0.10801	0.11786	onevsone	379.54	4.4506
4	Accept	0.49365	1411.8	0.10801	0.11291	onevsall	174.53	-
5	Accept	0.30864	49.647	0.10801	0.12218	onevsone	0.001092	0.001107
6	Accept	0.21342	48.628	0.10801	0.10803	onevsone	0.16683	3.2008
7	Accept	0.39043	1833.8	0.10801	0.10803	onevsone	999.24	-
8	Accept	0.30864	62.57	0.10801	0.10804	onevsone	984.41	0.02655
9	Best	0.089386	55.72	0.089386	0.089425	onevsone	200.4	9.9998
10	Best	0.08283	184.13	0.08283	0.082831	onevsone	943.73	181.64
11	Accept	0.13382	26.408	0.08283	0.082838	onevsone	17.595	610.08
12	Best	0.082102	145.97	0.082102	0.081875	onevsone	577.18	52.455
13	Accept	0.22914	31.989	0.082102	0.0819	onevsone	0.29729	160.36
14	Accept	0.31967	34.237	0.082102	0.081909	onevsone	0.0011939	0.001464
15	Best	0.081582	84.992	0.081582	0.08154	onevsone	239.14	62.504
16	Accept	0.11759	125.21	0.081582	0.081561	onevsall	883.04	796.61
17	Accept	0.31051	33.385	0.081582	0.081524	onevsall	0.42508	824.08
18	Accept	0.097919	40.447	0.081582	0.081503	onevsall	947.01	5.9315
19	Accept	0.30864	52.32	0.081582	0.081549	onevsall	964.98	0.007614
20	Best	0.080125	177.42	0.080125	0.080187	onevsall	710.6	65.12

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Coding	BoxConstraint	KernelScale
21	Accept	0.30864	48.69	0.080125	0.080185	onevsone	973.5	0.9980
22	Accept	0.086056	208.04	0.080125	0.081018	onevsall	931.41	30.686
23	Accept	0.087513	89.93	0.080125	0.081599	onevsall	162.09	29.174
24	Best	0.079813	216.59	0.079813	0.079772	onevsall	404.52	76.7
25	Accept	0.27097	1283.8	0.079813	0.07977	onevsone	0.0010156	-
26	Accept	0.12206	51.886	0.079813	0.079836	onevsone	774.88	974.12
27	Accept	0.091883	24.028	0.079813	0.079889	onevsone	29.161	24.276
28	Accept	0.13247	25.053	0.079813	0.079902	onevsall	884.76	215.85
29	Accept	0.20187	14.091	0.079813	0.07991	onevsall	0.85572	98.439
30	Accept	0.30864	30.242	0.079813	0.07993	onevsall	806.75	0.6371

Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 7691.7696 seconds
Total objective function evaluation time: 7656.435

Best observed feasible point:

Coding	BoxConstraint	KernelScale	KernelFunction	PolynomialOrder	Standardize
onevsall	404.52	76.7	gaussian	NaN	false

Observed objective function value = 0.079813
Estimated objective function value = 0.07993
Function evaluation time = 216.5894

Best estimated feasible point (according to models):

Coding	BoxConstraint	KernelScale	KernelFunction	PolynomialOrder	Standardize
onevsall	404.52	76.7	gaussian	NaN	false

```
Estimated objective function value = 0.07993
Estimated function evaluation time = 109.7561
Warning: Some output might be missing due to a network interruption. To get the missing output, rerun the
script.
```

Save the extracted features and model. Here "US" stands for Majority undersampling. While considering SMOTE, replace US by SMOTE.

```
save('C:\courses\ML\project\SVM_2_class\trained_model_SVM_US.mat','model');
save('C:\courses\ML\project\SVM_2_class\Training_features_SVM_US.mat','trainFeatures');
save('C:\courses\ML\project\SVM_2_class\Validation_features_SVM_US.mat','ValidationFeatures');
save('C:\courses\ML\project\SVM_2_class\Testing_features_SVM_US.mat','testFeatures');
save('C:\courses\ML\project\SVM_2_class\Trainingred_features_SVM_US.mat','trainFeatures1');
save('C:\courses\ML\project\SVM_2_class\Validationred_features_SVM_US.mat','ValidationFeatures1
save('C:\courses\ML\project\SVM_2_class\Testingred_features_SVM_US.mat','testFeatures1');
save('C:\courses\ML\project\SVM_2_class\TestingINDICES_features_SVM_US.mat','selidx');
save('C:\courses\ML\project\SVM_2_class\Training_featureweights_SVM_US.mat','train_reduced');
```

Support functions

```
function [trainFeatures,ValidationFeatures, testFeatures,featureindices] = helperExtractFeature
% This function is only in support of XpwWaveletMLExample. It may change or
% be removed in a future release.
trainFeatures = [];
testFeatures = [];
ValidationFeatures = [];
for idx =1:size(trainData,1)
    x = trainData{idx,:};
    x = detrend(x,0);
    arcoefs = blockAR(x,AR_order,T);
    se = shannonEntropy(x,T,level);
    [cp,rh] = leaders(x,T);
    wvar = modwtvar(modwt(x,'db2'),'db2');
    [mobility,complexity] = HjorthParameters(x);
    ifq = instfreq(x,200);
    trainFeatures = [trainFeatures; arcoefs se cp rh wvar' mobility complexity mean(ifq) mean(x

end

for idx =1:size(ValData,1)
    x1 = ValData{idx,:};
    x1 = detrend(x1,0);
    arcoefs = blockAR(x1,AR_order,T);
    se = shannonEntropy(x1,T,level);
    [cp,rh] = leaders(x1,T);
    wvar = modwtvar(modwt(x1,'db2'),'db2');
        [mobility,complexity] = HjorthParameters(x);
    ifq = instfreq(x,200);
    ValidationFeatures = [ValidationFeatures;arcoefs se cp rh wvar' mobility complexity mean(if

end
```

```
for idx =1:size(testData,1)
    x1 = testData{idx,:};
    x1 = detrend(x1,0);
    arcoefs = blockAR(x1,AR_order,T);
    se = shannonEntropy(x1,T,level);
    [cp,rh] = leaders(x1,T);
    wvar = modwtvar(modwt(x1,'db2'),'db2');
        [mobility,complexity] = HjorthParameters(x);
    ifq = instfreq(x,200);
    testFeatures = [testFeatures;arcoefs se cp rh wvar' mobility complexity mean(ifq) mean(x) v

end

featureindices = struct();
% 4*8
featureindices.ARfeatures = 1:32;
startidx = 33;
endidx = 33+(16*8)-1;
featureindices.SEfeatures = startidx:endidx;
startidx = endidx+1;
endidx = startidx+7;
featureindices.CP2features = startidx:endidx;
startidx = endidx+1;
endidx = startidx+7;
featureindices.HRfeatures = startidx:endidx;
startidx = endidx+1;
endidx = startidx+13;
featureindices.WVARfeatures = startidx:endidx;
end


function se = shannonEntropy(x,numbuffer,level)
numwindows = numel(x)/numbuffer;
y = buffer(x,numbuffer);
se = zeros(2^level,size(y,2));
for kk = 1:size(y,2)
    wpt = modwpt(y(:,kk),level);
    % Sum across time
    E = sum(wpt.^2,2);
    Pij = wpt.^2./E;
    % The following is eps(1)
    se(:,kk) = -sum(Pij.*log(Pij+eps),2);
end
se = reshape(se,2^level*numwindows,1);
se = se';
end


function arcfs = blockAR(x,order,numbuffer)
numwindows = numel(x)/numbuffer;
y = buffer(x,numbuffer);
arcfs = zeros(order,size(y,2));
for kk = 1:size(y,2)
```

```
    artmp =  arburg(y(:,kk),order);
    arcfs(:,kk) = artmp(2:end);
end
arcfs = reshape(arcfs,order*numwindows,1);
arcfs = arcfs';
end


function [cp,rh] = leaders(x,numbuffer)
y = buffer(x,numbuffer);
cp = zeros(1,size(y,2));
rh = zeros(1,size(y,2));
for kk = 1:size(y,2)
    [~,h,cptmp] = dwtleader(y(:,kk));
    cp(kk) = cptmp(2);
    rh(kk) = range(h);
end
end

function [mobility,complexity] = HjorthParameters(xV)

n = length(xV);
dxV = diff([0;xV]);
ddxV = diff([0;dxV]);
mx2 = mean(xV.^2);
mdx2 = mean(dxV.^2);
mddx2 = mean(ddxV.^2);

mob = mdx2 / mx2;
complexity = sqrt(mddx2 / mdx2 - mob);
mobility = sqrt(mob);
end
```