Supplementary Material. Source Code for All Steps Requiring MATLAB Processing

```
%Data Preprocessing: Isolating Samples & Converting Channels
%File name 'sddb' = SCA cohort data, 'nsrdb' = normal cohort data
filename1='sddb30_V1P';
N=2685096;
N0=2670006;
%Where 'P' denotes the preprocessed ECG (double channel recording)

eval(['[tm,sig_' filename1 ']=rdsamp(''sddb/30'',[],N,N0);']);
filename2 = strcat(filename1,'.mat');
varname = strcat('sig_',filename1);
save (filename2,varname);

%Converts recording from double channel to single channel
sddb30_V1 =sig_sddb30_V1P(:,1)-sig_sddb30_V1P(:,2);
save('sddb30_V1.mat','sddb30_V1');

%Data Preprocessing: Baseline Wander Removal
%Sampling rate = 250Hz
fs=250;

load('sddb30_V1.mat');
ecg = sddb30_V1(:); % convert to vector first (single channel)
ecg_raw = sddb30_V1; %take the raw signal for plotting later
time_scale = length(ecg_raw)/fs; % total time;
%Noise cancelation (Filtering)
f1=0.5; %cutoff low frequency to get rid of baseline wander
f2=45; %cutoff frequency to discard high frequency noise
Wn=[f1 f2]*2/fs; % cut off based on fs
N = 3; % order of 3 less processing
[a,b] = butter(N,Wn); %bandpass filtering
ecg = filtfilt(a,b,ecg);
save('sddb30_V1BWR.mat','ecg')

%Feature Extraction
%Load data (after baseline wander has been removed)
```

```
load('sddb41_V16BWR.mat');
fs=250; %Sampling frequency = 250Hz

% Detect R wave peak
[~,locs_Rwave] = findpeaks(ecg(1:250*20),'MinPeakHeight',0.5,...
                                    'MinPeakDistance',120);
%Calculate Heart Rate (bpm)
HR = length(locs_Rwave)*3

%Importing R-R Interval Related Features from Excel to MATLAB
clear all;

filename1='nsrdb19830_RR';    %change file name
%N.B: Change sheet with each patient record e.g. sddb31 = 2
num=xlsread('RR Interval.xlsx', 15,'D2:D61'); %change cell range
meanRR = strcat(filename1,'.mat');
save(meanRR, 'num');

filename2='nsrdb19830_RRD';
num1=xlsread('RR Interval.xlsx', 15, 'L2:L61');
diff = strcat(filename2,'.mat');
save(diff,'num1');
filename3='nsrdb19830_RRD2';
num2=xlsread('RR Interval.xlsx', 15, 'M2:M61');
diff_sq = strcat(filename3,'.mat');
save(diff_sq, 'num2');

%Combining the data into a matrix based on time frames prior to SCA for
classification processing
%SCA Patient Cohort
Subject_list = [30 31 32 34 36 41 45 46 51 52];

for i=1:length(Subject_list);
Subject_list(i)
%'V17' is the only section that changes
eval(['load ''sddb' num2str(Subject_list(i)) 'V17_RR.mat'';']);
```

```
eval(['load ''sddb' num2str(Subject_list(i)) 'V17_RRD.mat'';']);
eval(['load ''sddb' num2str(Subject_list(i)) 'V17_RRD2.mat'';']);
if i==1
    data1= horzcat(num,num1,num2);
    data2 = data1;
else
    data1= horzcat(num,num1,num2);
    data2 = vertcat(data2,data1);
end
end

save('SCA_3hrs.mat','data2');

%Normal Patient Cohort
Subject_list2 = [16265 16420 16483 16539 17052 19088 19090 19093 19140
19830];

for i=1:length(Subject_list2);
Subject_list2(i)
eval(['load ''nsrdb' num2str(Subject_list2(i)) '_RR.mat'';']);
eval(['load ''nsrdb' num2str(Subject_list2(i)) '_RRD.mat'';']);
eval(['load ''nsrdb' num2str(Subject_list2(i)) '_RRD2.mat'';']);
if i==1
    data3=horzcat(num,num1,num2);
    data4 = data3;
else
    data1=horzcat(num,num1,num2);
    data4=vertcat(data4,data1);
end
end

save('NORMAL.mat','data4');

%Organising the data for Supervised Machine Learning
load 'sca_input1min.mat'
load 'classes.mat'
```

```
input = input';

X=input;

Y=classes;

data=array2table(X);

data.Group=Y;


%Train and Classify patterns with a Linear Discriminant Classifier

function [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)

% Extract predictors and response

% This code processes the data into the right shape for training the

% classifier.

inputTable = trainingData;

predictorNames = {'X1', 'X2', 'X3'};

predictors = inputTable(:, predictorNames);

response = inputTable.Group;

isCategoricalPredictor = [false, false, false];


% Train a classifier

% This code specifies all the classifier options and trains the classifier.

classificationDiscriminant = fitcdiscr(...

    predictors, ...

    response, ...

    'DiscrimType', 'diagLinear', ...

    'FillCoeffs', 'off', ...

    'SaveMemory', 'on', ...

    'ClassNames', {'nsca'; 'sca'});


% Create the result struct with predict function

predictorExtractionFcn = @(t) t(:, predictorNames);

discriminantPredictFcn = @(x) predict(classificationDiscriminant, x);

trainedClassifier.predictFcn = @(x)
discriminantPredictFcn(predictorExtractionFcn(x));


% Add additional fields to the result struct

trainedClassifier.RequiredVariables = {'X1', 'X2', 'X3'};

trainedClassifier.ClassificationDiscriminant = classificationDiscriminant;
```

```
trainedClassifier.About = 'This struct is a trained classifier exported
from Classification Learner R2016a.';

trainedClassifier.HowToPredict = sprintf('To make predictions on a new
table, T, use: \n  yfit = c.predictFcn(T) \nreplacing ''c'' with the name
of the variable that is this struct, e.g. ''trainedClassifier''. \n \nThe
table, T, must contain the variables returned by: \n  c.RequiredVariables
\nVariable formats (e.g. matrix/vector, datatype) must match the original
training data. \nAdditional variables are ignored. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot, ''stats'',
''stats.map''), ''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');


% Extract predictors and response

% This code processes the data into the right shape for training the

% classifier.

inputTable = trainingData;

predictorNames = {'X1', 'X2', 'X3'};

predictors = inputTable(:, predictorNames);

response = inputTable.Group;

isCategoricalPredictor = [false, false, false];


% Perform cross-validation

partitionedModel = crossval(trainedClassifier.ClassificationDiscriminant,
'KFold', 5);


% Compute validation accuracy

validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');


% Compute validation predictions and scores

[validationPredictions, validationScores] = kfoldPredict(partitionedModel);


%Train and Classify patterns with a Linear Support Vector Machine
Classifier

function [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)

% Extract predictors and response

% This code processes the data into the right shape for training the

% classifier.

inputTable = trainingData;

predictorNames = {'X1', 'X2', 'X3'};
```

```
predictors = inputTable(:, predictorNames);
response = inputTable.Group;
isCategoricalPredictor = [false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationSVM = fitcsvm(...
    predictors, ...
    response, ...
    'KernelFunction', 'linear', ...
    'PolynomialOrder', [], ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', {'nsca'; 'sca'});

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'X1', 'X2', 'X3'};
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained classifier exported
from Classification Learner R2016a.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new
table, T, use: \n  yfit = c.predictFcn(T) \nreplacing ''c'' with the name
of the variable that is this struct, e.g. ''trainedClassifier''. \n \nThe
table, T, must contain the variables returned by: \n  c.RequiredVariables
\nVariable formats (e.g. matrix/vector, datatype) must match the original
training data. \nAdditional variables are ignored. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot, ''stats'',
''stats.map''), ''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
```

```
inputTable = trainingData;

predictorNames = {'X1', 'X2', 'X3'};

predictors = inputTable(:, predictorNames);

response = inputTable.Group;

isCategoricalPredictor = [false, false, false];


% Perform cross-validation

partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold',
5);


% Compute validation accuracy

validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');


% Compute validation predictions and scores

[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
```

%Train and Classify patterns with a Non-linear (Fine Gaussian) Support Vector Machine Classifier

```
function [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)

% Extract predictors and response

% This code processes the data into the right shape for training the

% classifier.

inputTable = trainingData;

predictorNames = {'X1', 'X2', 'X3'};

predictors = inputTable(:, predictorNames);

response = inputTable.Group;

isCategoricalPredictor = [false, false, false];


% Train a classifier

% This code specifies all the classifier options and trains the classifier.

classificationSVM = fitcsvm(...

    predictors, ...

    response, ...

    'KernelFunction', 'gaussian', ...
```

```
    'PolynomialOrder', [], ...
    'KernelScale', 0.43, ...
    'BoxConstraint', 1, ...
    'Standardize', true, ...
    'ClassNames', {'nsca'; 'sca'});


% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x)
svmPredictFcn(predictorExtractionFcn(x));


% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'X1', 'X2', 'X3'};
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained classifier exported
from Classification Learner R2016a.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new
table, T, use: \n  yfit = c.predictFcn(T) \nreplacing ''c'' with the name
of the variable that is this struct, e.g. ''trainedClassifier''. \n \nThe
table, T, must contain the variables returned by: \n  c.RequiredVariables
\nVariable formats (e.g. matrix/vector, datatype) must match the original
training data. \nAdditional variables are ignored. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot, ''stats'',
''stats.map''), ''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');


% Extract predictors and response
% This code processes the data into the right shape for training the
% classifier.
inputTable = trainingData;
predictorNames = {'X1', 'X2', 'X3'};
predictors = inputTable(:, predictorNames);
response = inputTable.Group;
isCategoricalPredictor = [false, false, false];


% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold',
5);
```

```
% Compute validation accuracy

validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun',
'ClassifError');


% Compute validation predictions and scores

[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
```