SHUDDHANK PUSHOLA ( S20170020239 )

# ASSIGNMENT 02
# CLASSIFICATION OF EMG DATA

## Summary

The objective of this assignment is to classify EMG data into two classes: 1) Aggressive Activities, 2) Normal Activities. Our aim is to find optimal feature vectors for the given time series EMG data and compare different classifiers.

The given dataset is a time series EMG data with 10000 samples (15 actions per experimental session) each for 8 channels (4 for arms, 4 for legs)

### Features Extracted

Since the 10000 samples are representing the same action being repeated 15 times so we take segments of length 10000/15 = 666 to compute the feature vectors

1) **Time Series Statistics :** For each segment for each channel we compute the mean, variance, skewness and kurtosis as the features.
2) **Inter Channel Statistics :** We compute the cross correlation between any two arm or leg channels and take its maximum value.
3) **Log Moments of Fourier Spectra (LMF) :** We calculate the L point fourier transform for the segment. We then square the absolute of the coefficients. We then calculate the ith frequency domain moments. Using that we compute the 7 moment features and the 10 pairwise moment features for each channel,
4) **Spectral Band Powers :** For each channel of the p-th pattern, assuming a model order ν, we calculate the power spectral density estimate and then take N bands and calculate the power for each band

## Classification

We classify the data into two classes

1) Aggressive Activities
2) Normal Activities

We test different classifiers for the given feature data and compare their accuracy.

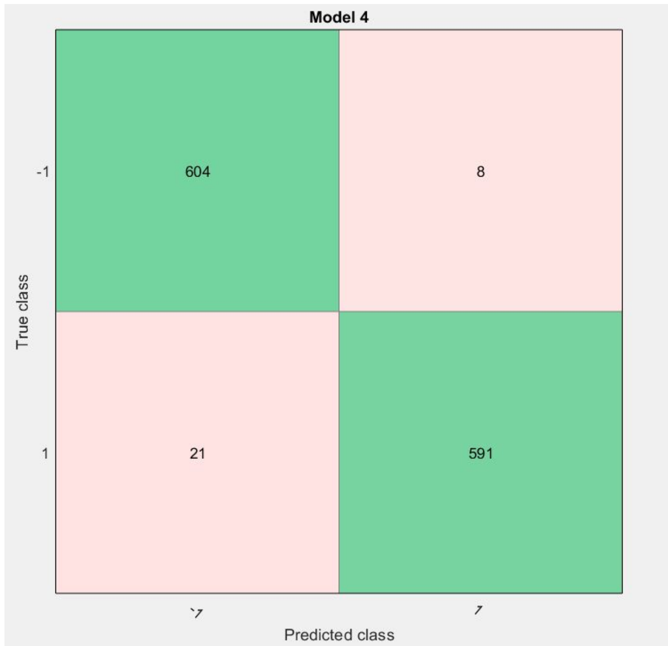
## Comparison of The Different Classifiers

| CLASSIFIER | ACCURACY |
|---|---|
| Linear SVM | 97.6% |
| Poly SVM | 99.4% |
| Gaussian SVM | 99.4% |
| KNN | 99.4% |
| Decision Trees | 100% |
| Bagged Trees | 100% |

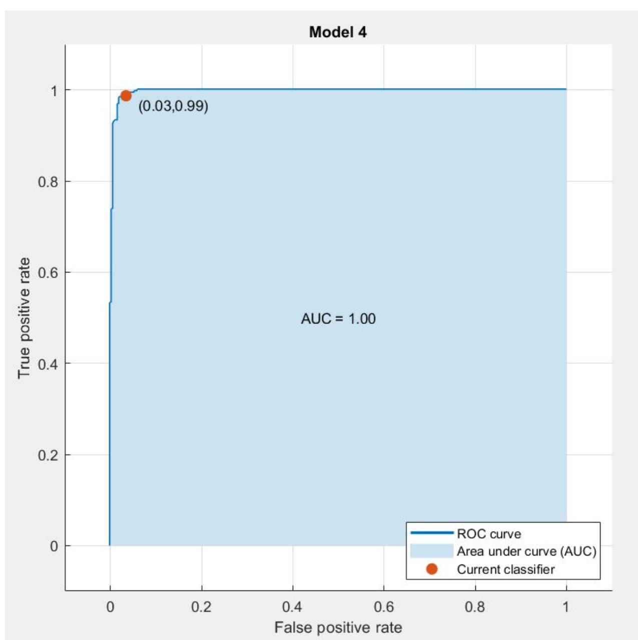
## Linear SVM

**Accuracy : 97.6%**

**Confusion Matrix:**



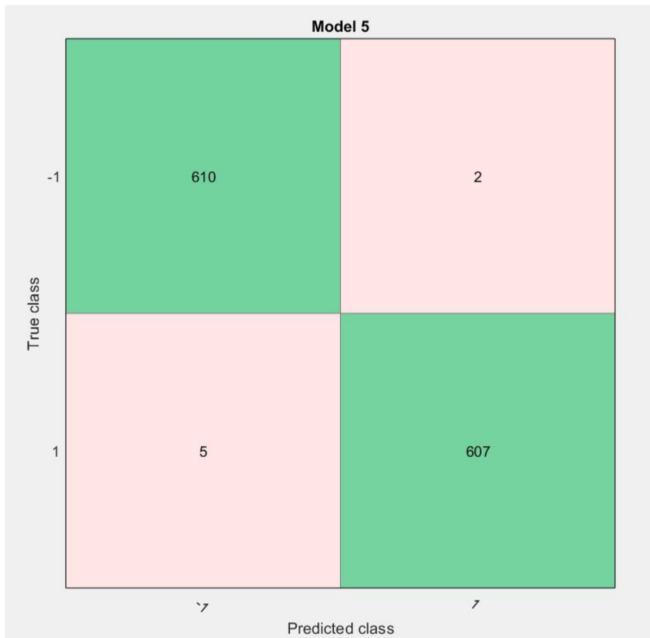

**ROC Curve:**



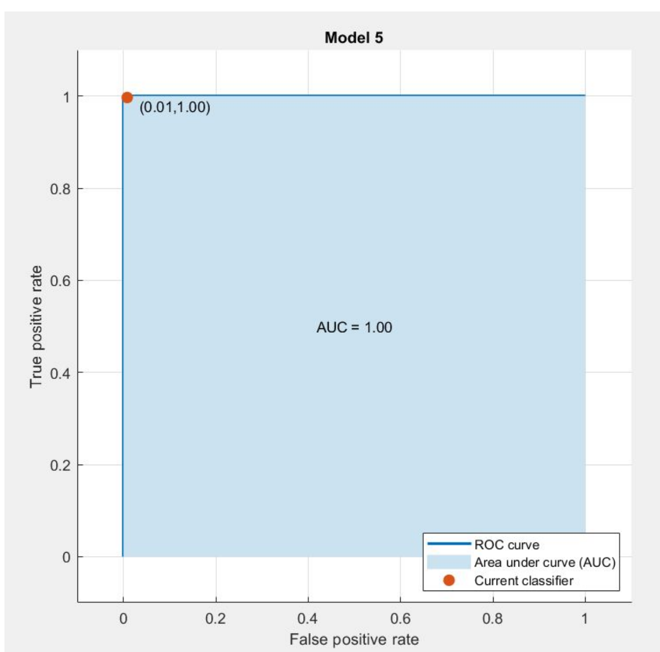
## Polynomial SVM

**Accuracy : 99.4%**

**Confusion Matrix:**




**ROC Curve:**

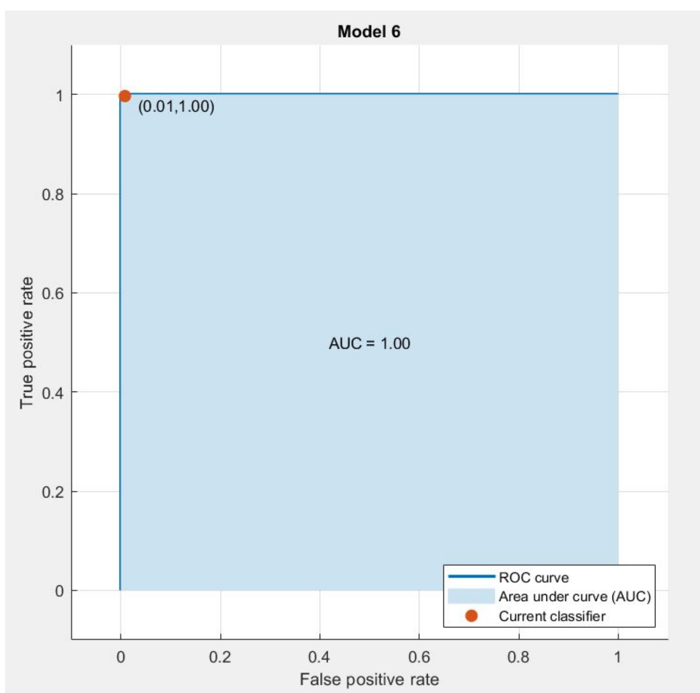


## Gaussian SVM

**Accuracy : 99.4%**

**Confusion Matrix:**

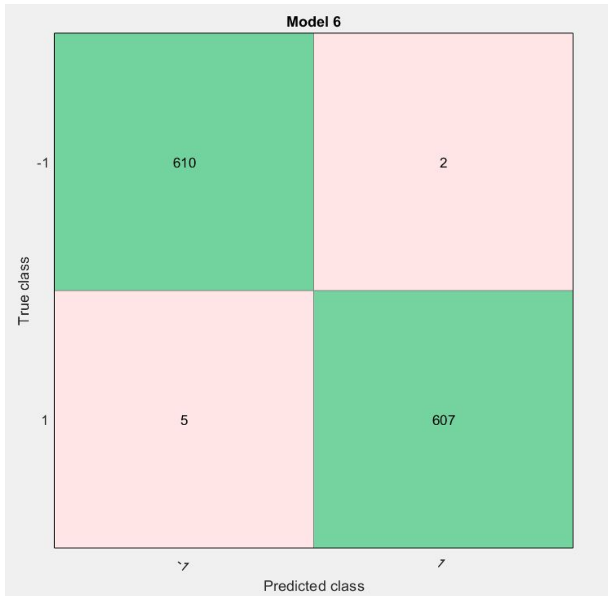

**ROC Curve:**



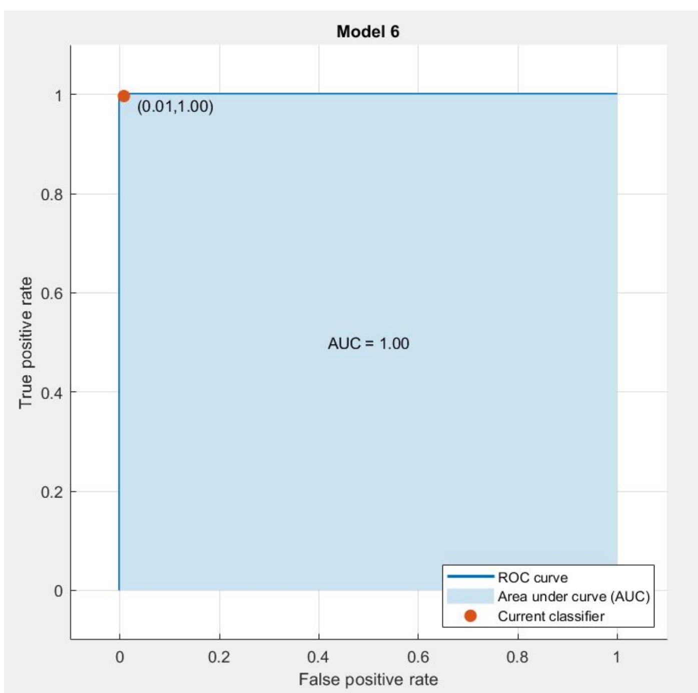
## K Nearest Neighbours

**Accuracy : 99.4%**

**Confusion Matrix:**



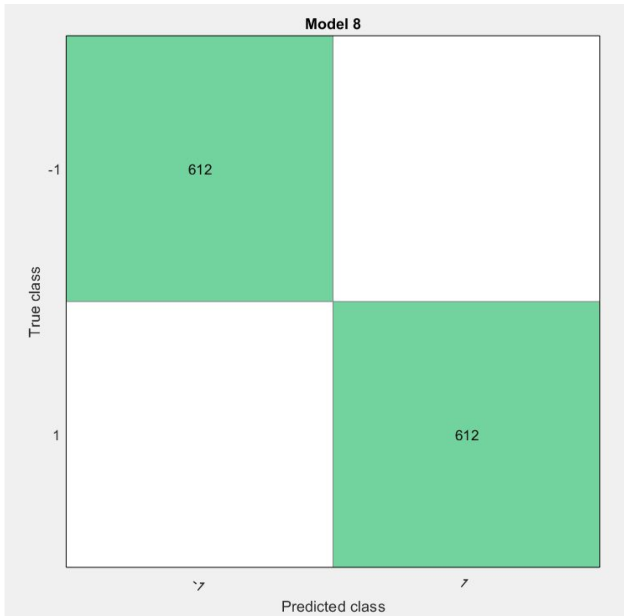

**ROC Curve:**



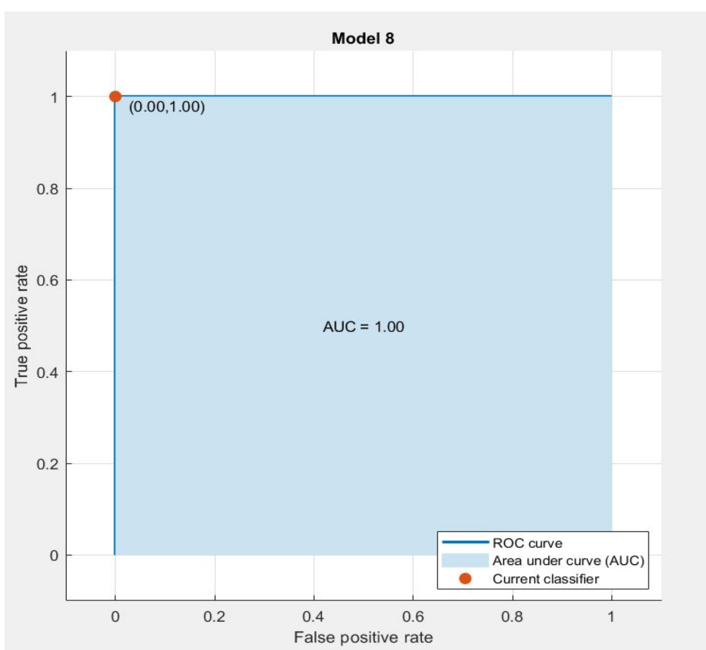
## Decision Tree

**Accuracy : 100%**

**Confusion Matrix:**



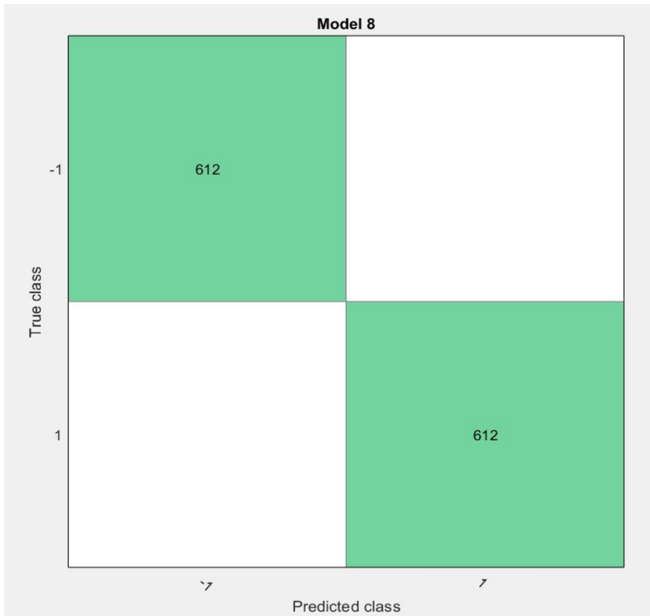

**ROC Curve:**



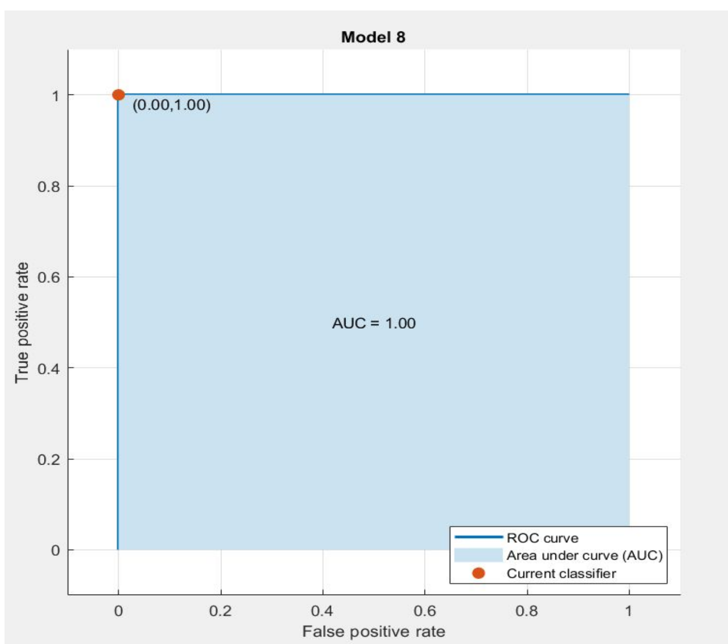
## Bagged Trees

**Accuracy : 100%**

**Confusion Matrix:**




**ROC Curve:**



## Code for Best Classifiers (Bagged Trees):

**Base Code :**

```
clc;
clear;
clear all;

a = ["sub1" , "sub2" , "sub3" , "sub4"];
b = ["Aggressive" , "Normal"];

data1 = [];
data2 = [];
feat1 = [];
feat2 = [];

for i = 1:4
    for j = 1:2
        for k = 1:10
            if j == 1
                file = strcat(a(i),"\",b(j),"\txt");
                files = dir(fullfile(file,"*.txt"));
                data11 = importdata(files(k).name);

                %Feature Extraction

                for n = 1:666: length(data11)
                    disp(k);
                    datax = data11(n:min((n+666),length(data11)),:);
                    f1 = time_fec(datax); % Time Series Statistics Features
                    f2 = lmf_moment(datax); %LMF
                    f3 = ics_fec(datax); %Cross Correlation
```

```
            f4 = sbp_burg(datax,2); %SPB using burg's method
            ff = [f1 f2 f3 f4 1];
            feat1 = [feat1' ff'];
            feat1 = feat1';
          end
        end
        if j == 2
          file = strcat(a(i),"\",b(j),"\txt");
          files = dir(fullfile(file,"*.txt"));
          data22 = importdata(files(k).name);
          for n = 1:666: length(data22)
            disp(k);
            datax = data22(n:min((n+666),length(data22)),:);
            f1 = time_fec(datax);
            f2 = lmf_moment(datax);
            f3 = ics_fec(datax);
            f4 = sbp_burg(datax,2);
            ff = [f1 f2 f3 f4 -1];
            feat2 = [feat2' ff'];
            feat2 = feat2';
          end
        end
      end
    end
  end

N = min(length(feat1'),length(feat2'));
trainingdata(1:2:2*N-1,:) = feat1(1:N,:);
trainingdata(2:2:2*N,:) = feat2(1:N,:);

[classifier validacc] = trainClassifier(trainingdata);

partitionedModel = crossval(classifier.ClassificationEnsemble, 'KFold', 5);
```

```
% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

confmat = confustionmat(trainingdata(:,261),validationPredictions);
```

**Classifier Code :**

***In certain lines the code for input table entries etc was too long so to minimize it I have represented it as "…………………………"**

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

inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', ………………………………………………………………………………………'column_260', 'column_261'});


predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', 'column_7', 'column_8', 'column_9', 'column_10', 'column_11', 'column_12', 'column_13', ……………………………………………………………………………'column_257', 'column_258', 'column_259', 'column_260'};

predictors = inputTable(:, predictorNames);

response = inputTable.column_261;

isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false, false, false, ………………………………………………………………………………, false];
```

```
% Train a classifier
% This code specifies all the classifier options and trains the classifier.
template = templateTree(...
    'MaxNumSplits', 1223);
classificationEnsemble = fitcensemble(...
    predictors, ...
    response, ...
    'Method', 'Bag', ...
    'NumLearningCycles', 30, ...
    'Learners', template, ...
    'ClassNames', [-1; 1]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.ClassificationEnsemble = classificationEnsemble;
% Extract predictors and response
```

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

% model.

% Convert input to table

inputTable = array2table(trainingData, 'VariableNames', {'column_1', 'column_2', 'column_3', 'column_4', 'column_5',.............................................................................'column_257', 'column_258', 'column_259', 'column_260', 'column_261'});


predictorNames = {'column_1', 'column_2', 'column_3', 'column_4', 'column_5', 'column_6', ......................................................'column_254', 'column_255', 'column_256', 'column_257', 'column_258', 'column_259', 'column_260'};

predictors = inputTable(:, predictorNames);

response = inputTable.column_261;

isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false, false,................, false];

%....... Represents repeating data

% Perform cross-validation

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


% Compute validation predictions

[validationPredictions, validationScores] = kfoldPredict(partitionedModel);


% Compute validation accuracy

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