CS470 Semester Project

[1]Fahad Fareed, [2]Muhammad Abdullah Ahsan Khan

SEECS, NUST
Islamabad, Pakistan.
[1]13beeffareed@seecs.edu.pk
[2]13beemahsan@seecs.edu.pk

***Abstract*— The artifacts introduced by physical activity in the captured data of an electrical impedance plethysmographic sensor can be used as features for detecting the physical activity. Up till now these artifacts were considered as a nuisance in the EIP sensor data. Therefore the infected data would most often be discarded. But [1] proposes a novel way of using this artifacts data to detect physical activity without the need of hardware sensors such as accelerometers. In this paper we proposed a technique for the classification of the sample wavelet signatures into five distinct physical activities (coughing, walking, eating, reaching and rolling in bed). The classification is done on the features designed specifically to capture the time-frequency signatures of the physical activities. The classifier used was an SVM with different types of kernels (linear, quadratic and cubic).**

***Keywords*—**

I. Introduction

There is a need for simple and cost effective real time health monitoring systems in the health industry. These systems are capable of tracking different patient health metrics even in non-clinical settings. In order to make them cost effective a single sensor output can be used to make multiple health metric inferences. In this way, these monitoring systems not only save human life but are also financially implementable. Such systems are abbreviated as RHMSs.

A good and reliable RHMS monitors a patient's different conditions. These conditions are then treated as features in order to correctly define/classify illness that a patient might have. These features include different physical, physiological, mental, and environmental conditions.

Electrical Impedance Plethysmographs or EIPs are a non-invasive method for tracking a patients vitals like heart-rate, respiratory rate, etc. The signal obtained from these sensors is distorted whenever the patient moves or is undergoing some physical activity. This distortion or motion artifacts are generally undesirable in the signal and are mostly the cause of data disposal. But a different approach can be taken to these artifacts. These artifacts can be used as a feature in order to detect and classify the type of physical activity that the patient might be undergoing. After the classification, a standard model could also be used to detect and classify patient illness.

In this paper a technique is presented that is able to detect and classify physical activity that a patient is performing using the EIP motion artifacts as the features.

II. Theory

a. *Cross-validation*

Cross validation is a model validation technique used to assess the accuracy of different trained data sets in practice. It is used in settings where one wants to predict a data sample using trained classifiers. A model is usually given known data and unknown data sets and the model uses known data to train itself whereas unknown dataset is used to test the model.

The data is partitioned into different folds depending on the figure 'n' taken and the 'n-1' folds are used to train data whereas the 'nth' fold is used for testing. In this way the model is analyzed in training phase in order to limit overfitting and giving insight to how the model will work in general on independent data sets. The cross validation is used instead of the conventional validation due to not enough availability of data to separate into training and testing sets without significant lose.

b. SVM

Support vector machine is a supervised learning model with associated algorithms that are used to train the model in order to classify the data into different classes depending on the type of features. An SVM model is a representation of points in space where points from each class are separated from points of other classes using a classifying boundary that is as wide as possible in gap size. Data sets are mapped into SVM models that predict the classes of the data points depending on which side of the boundary they fall. The classification boundary is also called hyperplane.

Consider training data points in following format:

$$(\vec{x_1}, y_1), (\vec{x_2}, y_2), ..., (\vec{x_n}, y_n),$$

Where y is the class of the data point x.

$$\vec{w}.\vec{x} - b = 0,$$

The w is normal to the hyperplane which we are supposed to maximize using equation:

$$\frac{1}{n}\sum_{1\in n} max(0, 1 - y_i(\vec{w}.\vec{x_i} - b)) + \lambda.\vec{w}^2,$$

In addition to linear classification, SVM are also able to perform nonlinear classifications using the kernel trick which involves mapping the inputs into high dimensional feature space. Some of those SVMs are trained and tested on the given data set and the results are compared in order to decide which kernel is the best for classifying data on the basis of classification accuracy.

III. Methodology

The solution approach that we used is explained as follows. We start with pre-processed data that is in the form of extracted wavelet based signatures. We assume that the wavelet score curve for the normal and accelerated breathing for each activity session are already supplied. After getting the data of different lengths in time, all of the samples are made of equal length. Then the corresponding data samples from the normal and the accelerated breathings are concatenated together. After concatenation, the data is divided into four equal parts. Out of these four parts, three are used for training purposes while the remaining one part is utilized in the testing phase. The data is then passed onto the SVM model. SVMs with different kernels were used in order to make a comparison of accuracy. After this step, the classification step is performed which gives the numerical value of the accuracy achieved and the confusion matrix that is used to infer the BER.

The whole process is shown in the following flow diagram.

a. Preprocessing

In preprocessing, we shape the data into matrix form to achieve desired input form for SVM. For that reason, two operations are applied on the data sets. i) The data sets are given equal length by discarding first and last few samples. This is done by finding the midpoint of the data set and a number 'n<total samples/2' of samples are added before and after the midpoint. After that, respective HiCurve and LoCurve are concatenated and transformed into matrix form using MATLAB built-in functions. Now we have an input to train and test SVM's.

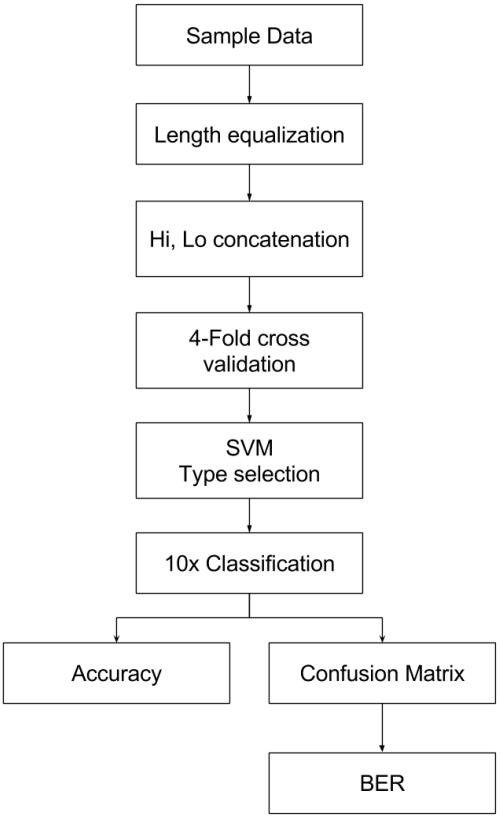

Fig 1: Classification Flow Diagram

b. SVM

The support vector machines tested were of 5 types; linear, quadratic, fine gaussian, medium gaussian, coarse Gaussian. Each type of Kernel was analysed and the one with the best results was chosen. The steps are as follows:

- Initialize Classification Learner from Matlab.
- Split the data into four folds. Generate a function from Matlab toolbox and use it to train and test on the data folds. Train the SVM on three of them while test on the remaining fold.
- Repeat this 10 times and measure the average accuracy.

- Run this process for each of the SVM Kernel and compare the results.
- Compute the confusion matrix Aij and calculate the balanced error rate (BER) represented by the formula:

$$BER = \frac{1}{M} \sum_{i \in M} \frac{(\sum_{j \in M} A_{ij}) - A_{ii}}{\sum_{j \in M} A_{ij}},$$

IV. Results

Kernel Type	Accuracy(%)	BER
Linear	82.40	0.0372
Quadratic	81.63	0.0372
Cubic	79.53	.0611
Fine Gaussian	37.21	.0654
Medium Gaussian	75.58	0.0732
Coarse Gaussian	57.75	0.4688

Table 1. Performance Metrics with change in kernel type

Order	Accuracy(%)	BER
Linear	82.40	0.0372
Quadratic	81.63	0.0372
Cubic	79.53	.0611
Fourth Order	71.11	.315
Fifth Order	68.44	0.632
Sixth Order	51.92	0.4144

Table 2. Performance metrics with change in polynomial order

The SVM algorithm is applied to the data taken from number of volunteers and results for accuracy and BER are taken against Kernel type and Order of the polynomial. In both cases the Linear SVM classifier provided the highest accuracy and lowest BER.

In Table 1. testing is done with five different Kernels. The linear SVM gives the best accuracy of 82.4% and BER of 0.0372. Quad Kernel gives almost equivalent results of 81.63% & 0.0372 whereas fine gaussian, medium gaussian & coarse gaussian significantly fall in accuracy with increase in BER.

In order to check whether increasing the polynomial would rather give better performances, the SVM were trained on higher polynomial order and accuracies were analyzed. Table 2. shows such performance with increase in order. The trend of the results is same in nature as previous one. Increasing the polynomial order decreases the accuracy performance meanwhile BER increases.

V. Discussion

The results taken from the simulations run on matlab classification learner tool show that linear SVM is the best classifier among all to classify the data with best accuracy. This was confirmed by running 10x each of the SVM classifier using the technique of cross validation with 4 folds. The linear classifier worked best due to the reason that a lot of data was present and overfitting was avoided using linear hyperplane.

For the sake of getting better SVM classifier, data sets were trained on higher number of polynomial orders. The results, however, were not satisfactory for nonlinear SVM models and simply decreased the performance of the system. The reason for that is simple, the higher order polynomials caused overfitting on the hyperspace boundary and due to large set of data present, the classification performance decreased with increase in order. The verdict is that linear SVM is best for current given data set classification among all kinds of SVM's and it produces the highest accuracy using linear hyperspace.

VI. Conclusion

The paper presented the opportunity to classify physical human activities using features from respiration sensor with the help of Support Vector Machines. We analyze the results of multiple SVM's and identify the best SVM Kernel for classifying the feature data. The results provide linear SVM being the classifier with the accuracy of 82.4% and BER of .0372.

VII. Matlab Code (Appendix)

A. Pre Processor

```
load('ProjectData.mat');
t1=HiCurve';  %assigning address of HiCurve vector array to a temporary variable
t2=LoCurve';  %assigning address of LoCurve vector array to a temporary variable
for k = 1 : length(t1)
   cellC = t1{k};
   mid=floor(length(t1{k})/2); %extracting the midpoint of the activity recorded
    t1{k} = cellC(mid-2700:mid+2500); %only keeping 4900 samples
end
for k = 1 : length(t2)
   cellC = t2{k};
   mid=floor(length(t2{k})/2); %extracting the midpoint of the activity recorded
    t2{k} = cellC(mid-2450:mid+2450); %only keeping 4900 samples
end
HiCurve=t1;  %restoring the vectors in HiCurve
LoCurve=t2; %restoring the vectors in LoCurve

features=cat(2,HiCurve,LoCurve);  %concatenating the array of vectors into a single vector
data = cell2mat(features);
save('PreprocessedData.mat', 'features');
```

B. Linear Train Classifier

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


% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 4);

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

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

%Confusion Matrix
predict = trainedClassifier.predictFcn(trainingData(:, 1:length(trainingData)-1));
response = trainingData(:, length(trainingData));
conf = confusionmat(response, predict);

%BER Rate
BER = 0;
for i = 1:length(conf)
        BER = BER + (sum(conf(i, :)) - conf(i, i))/sum(conf(i, :));
end
BER = BER/length(conf);
```

C. Train Classifier Coarse

```
function [trainedClassifier, validationAccuracy, conf, BER] = trainClassifier_coarse(trainingData)

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 4);

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

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

%Confusion Matrix
predict = trainedClassifier.predictFcn(trainingData(:, 1:length(trainingData)-1));
response = trainingData(:, length(trainingData));
conf = confusionmat(response, predict);

%BER Rate
BER = 0;
for i = 1:length(conf)
        BER = BER + (sum(conf(i, :)) - conf(i, i))/sum(conf(i, :));
end
BER = BER/length(conf);
```

D. Train Classifier Cube

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


% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 4);

% Compute validation predictions
[validationPredictions, validationScores] =
```

```
kfoldPredict(partitionedModel);

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

%Confusion Matrix
predict = trainedClassifier.predictFcn(trainingData(:,
1:length(trainingData)-1));
response = trainingData(:, length(trainingData));
conf = confusionmat(response, predict);

%BER Rate
BER = 0;
for i = 1:length(conf)
        BER = BER + (sum(conf(i, :)) - conf(i,
i))/sum(conf(i, :));
end
BER = BER/length(conf);
```

E. Train Classifier Fine

```
function [trainedClassifier, validationAccuracy, conf,
BER] = trainClassifier_fine(trainingData)


% Perform cross-validation
partitionedModel =
crossval(trainedClassifier.ClassificationSVM, 'KFold',
4);

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

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

%Confusion Matrix
predict = trainedClassifier.predictFcn(trainingData(:,
1:length(trainingData)-1));
response = trainingData(:, length(trainingData));
conf = confusionmat(response, predict);

%BER Rate
BER = 0;
for i = 1:length(conf)
        BER = BER + (sum(conf(i, :)) - conf(i,
i))/sum(conf(i, :));
end
BER = BER/length(conf);
```

F. Train Classifier Medium

```
function [trainedClassifier, validationAccuracy, conf,
BER] = trainClassifier_medium(trainingData)


% Perform cross-validation
partitionedModel =
crossval(trainedClassifier.ClassificationSVM, 'KFold',
4);

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

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

%Confusion Matrix
predict = trainedClassifier.predictFcn(trainingData(:,
1:length(trainingData)-1));
response = trainingData(:, length(trainingData));
conf = confusionmat(response, predict);

%BER Rate
BER = 0;
for i = 1:length(conf)
        BER = BER + (sum(conf(i, :)) - conf(i,
i))/sum(conf(i, :));
end
BER = BER/length(conf);
```

G. Train Classifier Quadratic

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


% Perform cross-validation
partitionedModel =
crossval(trainedClassifier.ClassificationSVM, 'KFold',
4);

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

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

%Confusion Matrix
predict = trainedClassifier.predictFcn(trainingData(:,
```

```
1:length(trainingData)-1));
response = trainingData(:, length(trainingData));
conf = confusionmat(response, predict);

%BER Rate
BER = 0;
for i = 1:length(conf)
        BER = BER + (sum(conf(i, :)) - conf(i, i))/sum(conf(i, :));
end
BER = BER/length(conf);
```

H. Training and Evaluation

```
accuracy = 0;
confmat = zeros(5, 5);
ber = 0;

for i = 1:10
    [trainedClassifier, validationAccuracy, conf, BER] = trainClassifier_quad(data);
   accuracy = accuracy + validationAccuracy;
   confmat = confmat + conf;
   ber = ber + BER;
end

accuracy = accuracy/10;
confmat = confmat./10;
ber = ber/10;
```

References

[1] Hassan Aqeel Khan, Amit Gore, Jeff Ashe and Shantanu Chakrabartty, *"Physical activity classification using time-frequency signatures of motion artifacts in multi-channel electrical impedance plethysmographs."*