Physical activity classification using time-frequency signatures of motion artifacts

Muhammad Mubeen School of Electrical Engineering and Computer Sciences (SEECS)
National University for Sciences and Technology
Islamabad, Pakistan
Email: 14beemmubeen@seecs.edu.pk Haroon Rashid School of Electrical Engineering and Computer Sciences (SEECS)
National University for Sciences and Technology
Islamabad, Pakistan
Email: 14beehrashid@seecs.edu.pk

***Abstract*—Physical Activities that human perform are know to cause motion artifacts. These motion artifacts can be sensed and consequently then measured using an Electrical Impedance Plethysmographic (EIP) sensor. These time-frequency artifacts can then be plotted as a time varying graph. It has been observed that each specific human action has in turn a unique motion artifact which when plotted against time can be seen clearly. Earlier these time frequency artifacts were discarded but recently due to the stated observation, we can apply various modern Machine Learning based techniques using which we can predict which motion artifact corresponded to which physical activity. The aim of this report is to analyze the time frequency artifacts obtained using the EIP sensor and apply various Machine Learning Classifiers and analyze their accuracy. A total of 5 different Machine Learning based classifiers were used including Linear SVM, Multi-Layer Perceptrons, Naive Bayes, Decision Trees and k-Nearest Neighbour Classifier. It turned out that on multiple trials the k-Nearest Neighbour classifier with k=3 won with an accuracy of about 82% and outperformed all the other classifier.**

I. Introduction

The output produced by an EIP sensor is a motion artifact. Our aim of this report is to examine them and then try to exploit these artifacts in order to predict the kind of physical activity which produced this artifact. It has been observed that each physical activity has its own unique time frequency sequence of the artifact. So, in order to do so, first of all data was collected using EIP sensors from different users. In total 19 adult human test subjects were used to collect the data. A total 10 EIP sensors were the body of each subject and then for each human subject a dataset almost 32 minutes was collected which included all of the physical activities. A sequence of events for 3 different recording sessions is shown in Fig. 1.

Each of the session started with the subject first being in the rest and breathing normally, after almost 30 seconds the subject performed some action/activity for a duration of about 60 seconds and then in the end again the test subject would sit in rest position for about 30 seconds. So, in short each session was about 2 minutes long. But in our case for the classification problem, only 129 samples were provided. These sample were further divided into two parts: HiCurves which corresponded to accelerated breathing and the other one was LoCurve which corresponded to normal breathing sequence. A simple comparision between the two is shown in the following figure:- Here, the dotted graph represented accelerated breathing sequence, whereas the normal one represented the normal breathing sequence. In addition to the samples, there class labels were also provided. These labels ranged from 1 to 5 and they in turn corresponded to some physical activity.

from 2017-06-19 00-07-05.png

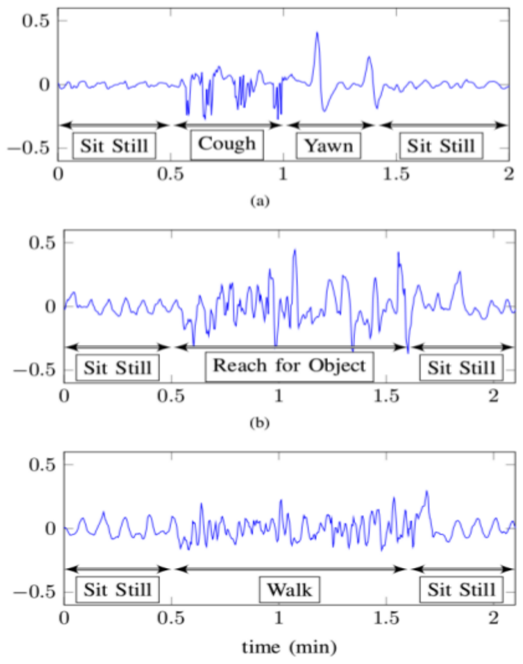


Fig. 1. Time Sequence of motion artifacts of 3 different physical actions

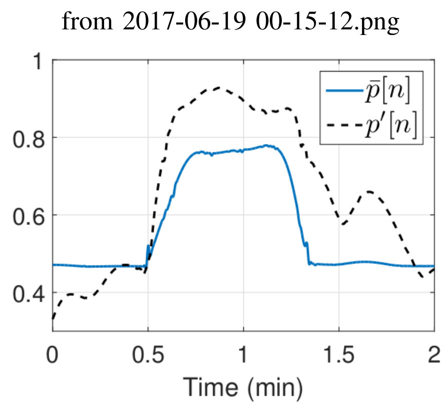


Fig. 2. wavelet score curves for accelerated and normal breathing

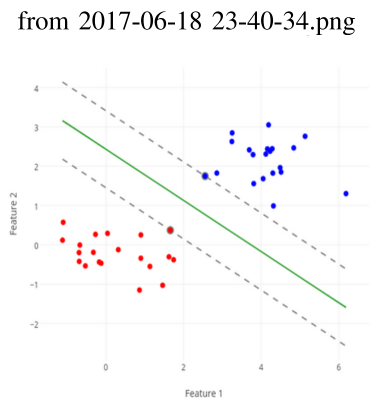


Fig. 3. A Linear SVM

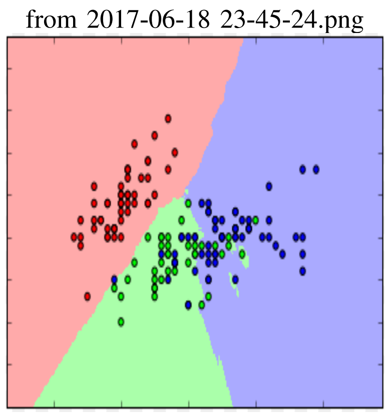


Fig. 4. A k-NN Classifier

II. Pre-Processing

In order for the classifier to work, some preprocessing had to be done in order to convert the given HiCurves and the LoCurves into valid feature vector. This was done in a two step procedure. First of all the feature vectors were not equi-length. So in order to make them equi-length we could discard a few samples at the start or at the end. In our case, we chose only the last 5500 elements of both the HiCurves and the LoCurves curves. After that, we concatenated the vectors HiCurve and LoCurve to form just a single feature vector which was to be fed into the classifier. The length of each vector was 1x11000 and we had 129 such vectors to play with. In the later step, the vectors were used for training the various classifiers, which have been discussed in the next section.

III. Classifiers

A total of 5 different Machine Learning classifiers were used for the classification of the activity patterns. They are discussed below

A. Support Vector Machine

The first classifier to be used was a Support Vector Machine, which is a maximum margin classifier. The kernel used for the classification was a linear one. Even though there are other kernels like radial etc. too but there accuracy turned out to be pretty low. A typical support vector machine is shown in the following figure.

B. K-Nearest Neighbour

In k-Nearest Neighbour, the object is classified by the majority vote of its neighbors. The object is assigned to the class which is most common among its k nearest neighbors. Here k is a hyper-parameter and in our case, we choose k=3 after several testing.

C. Decision Trees

The third classifier to be used is the Decision Trees. It is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs and utility.

D. Naive Bayes Classifier

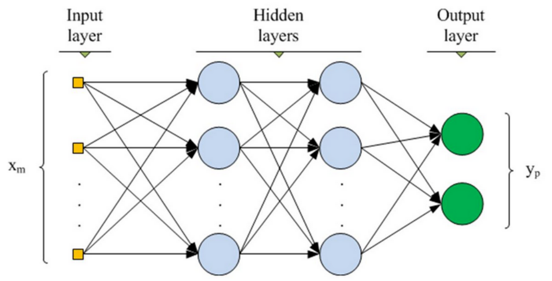
The forth classifier to be used was the Naive Bayes. It is a set of supervised learning algorithm based on applying the Bayes' theorem with a simple naive assumption of independence between every pair of features.

E. Multi-Layer Perceptions

The fifth and final classifier to be used was a multi-layer perceptron or MLPs. It is a feed-forward artificial neural network (ANN) model which maps the input data onto a set of appropriate outputs. In our case we used two hidden layers with sizes 7 and 4 respectively. The weight optimizer was chosen to be LBFGHS. The intial learning rate was 10ˆ-3 and kept constant throughout the training while the activation function i.e. non-linearity used was ReLU. The MLP turned out to be the most interesting of the the classifiers. The reason of course will be discussed later on in the next section.

IV. Performance Measurement Criteria

A total of 3 different criterion were considered for evaluation of the classifiers on the given dataset. They are given below:-



SVM.png

Fig. 5. An Artificial Neural Network (ANN) Classifier

A. Accuracy

1) Accuracy with Non Overlapping Folds: In order to find this, the data was first split, randomly in to 4 non-overlapping folds/partitions. Training of data was done on 3 folds and while testing on the 4t̂h fold. And this was repeated until all four folds have been in the test set one-by-one. Then in the end we average each one of the calculated accuracy over the 4-folds and stored it.

2) Accuracy with Overlapping Folds: In this part, the earlier steps were repeated but the only difference was that the 4-fold cross-validation was done with randomly created partitions at least 10 times and then in the end it was averaged.

B. Balanced Error Rate (BER)

The activity sessions provided were some what unbalanced therefore, accuracy is not a good metric to measure how good the classifier is performing. So, in addition to accuracy, another metric is used called the Balanced-Error-Rate (BER), which is more suitable for unbalanced data. It is simply the average of the errors on each class. It can be found using: -

$$BER = \frac{1}{M}\sum_{i=1}^{M}\frac{\sum_{i=1}^{M} A_{ij} - A_{ii}}{\sum_{i=1}^{M} A_{ij}}$$

where, A_{ij} denotes the element of the confusion matrix A at row-i and column-j

V. Results

The various results achieved of the performance metrics using different classifiers are summarized below in table 1.

From the above table, we can clearly see that a simple k-Nearest Neighbour has outperformed all state of the art techniques including the SVMs and the MLPs. The highest accuracy when no overlap in taken between the folds is in the case of k-NN, whereas when the folds/partitions are chosen randomly, the SVM gives the best accuracy. Similarly, the lowest BER (i.e. lowest per class error rate) also turns out to be in the case of Linear SVM whereas the second best turns out to be the Naive Bayes Algorithm. The one in the list turn out to be a bit of a shock is the MLPs as it is the state of the art technique. It has to be pointed out here that MLPs turned out to be the most inconsistent ones as the accuracy varied during each of the test. Interestingly, it turned out to be as high as 84% once also.

Classifier Used	Accuracy for 4-Folds (Non Overlapping)	Accuracy for 4 Folds (Overlapping)	Balanced Error Rate (BER)
Gaussian Naive Bayes	76%	75%	0.237
k-Nearest Neighbour	82%	77%	0.237
Multi-Layered Perceptrons	61%	65%	0.424
Decision Tree	60%	59%	0.476
Linear SVM	79%	81%	0.187

TABLE I. Accuracies and BER for various Classifiers

VI. Conclusion

To classify physical activities as presented in the above sections, more classical methods like linear SVM tends to perform much better than Multilayer perceptron or data driven deep learning approaches. SVM perform exceptionally well for physical activities classification both on accuracy measure as well as BER. Experiments run under this problem supports the idea that SVM tends to perform much better when data is small.

Appendix A
Python Code

```
import numpy as np
import scipy.io as sio
from sklearn.model_selection import
    cross_val_score

### import the sklearn modules for
    Different Classifiers
from sklearn.svm import SVC
from sklearn.neighbors import
    KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.neural_network import
    MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import
    confusion_matrix

# Getting the BER
def get_BER(y_pred, y_true, M=5):
    temp = 0
    BER = 0

    #Getting the Confusion Matrix
    conf_matrix = confusion_matrix(y_true,
        y_pred, labels=[1,2,3,4,5])
    # print conf_matrix
    # print y_pred
    # print y_true.ravel()
    # Solving the BER
    for i in range(0,M):
        for j in range(0,M):
            temp = temp + conf_matrix[i][j
                ]
        # Sometimes an entire row of
            matrix is zero, result in
            division by 0
        if temp == 0:
            continue
        BER = BER + (float(temp-
            conf_matrix[i][i])/temp)
        temp = 0

    return BER/M

#Support Vector Classifier to classify
    data
def classify_svm(features_train,
    labels_train, features_test,
    labels_test):
    #create classifier
    clf = SVC(kernel='linear')
    #fit or train the classifier on the
        training features and labels
    clf.fit(features_train,labels_train.
        ravel())
    #use the trained classifier to predict
         labels for the test features
    pred = clf.predict(features_test)
    #accuracy_scores calculates accuracy
        by comparing predicted data and
        test labels element by element
    accuracy = accuracy_score(labels_test,
        pred)
    return accuracy,pred

# k-nearest Neighbour Classifier to
    classify data
def classify_nn(features_train,
    labels_train, features_test,
    labels_test):
    #create classifier
    clf = KNeighborsClassifier(n_neighbors
        =3)
    #fit or train the classifier on the
        training features and labels
    clf.fit(features_train,labels_train.
        ravel())
    #use the trained classifier to predict
         labels for the test features
    pred = clf.predict(features_test)
    #accuracy_scores calculates accuracy
        by comparing predicted data and
        test labels element by element
    accuracy = accuracy_score(labels_test,
        pred)
    return accuracy,pred

#Guassian naive Bayes Classifier to
    classify data
def classify_nb(features_train,
    labels_train, features_test,
    labels_test):
    #create classifier
    clf = GaussianNB()
    #fit or train the classifier on the
        training features and labels
    clf.fit(features_train,labels_train.
        ravel())
    #use the trained classifier to predict
         labels for the test features
    pred = clf.predict(features_test)
    #accuracy_scores calculates accuracy
        by comparing predicted data and
        test labels element by element
    accuracy = accuracy_score(labels_test,
        pred)
    return accuracy,pred

# Decision Classifier to classify data
def classify_dc(features_train,
    labels_train, features_test,
    labels_test):
    #create classifier
    clf = tree.DecisionTreeClassifier()
    #fit or train the classifier on the
        training features and labels
    clf.fit(features_train,labels_train)
    #use the trained classifier to predict
         labels for the test features
    pred = clf.predict(features_test)
    #accuracy_scores calculates accuracy
        by comparing predicted data and
        test labels element by element
    accuracy = accuracy_score(labels_test,
        pred)
    return accuracy,pred

# Multi Layer PerceptronClassifier to
    classify data
def classify_mlp(features_train,
    labels_train, features_test,
    labels_test):
    #create classifier
    clf = MLPClassifier(solver='lbfgs',
        alpha=1e-3,hidden_layer_sizes
        =(7,4),activation='relu')
    #fit or train the classifier on the
        training features and labels
    clf.fit(features_train,labels_train.
        ravel())
    #use the trained classifier to predict
         labels for the test features
    pred = clf.predict(features_test)
    #accuracy_scores calculates accuracy
       by comparing predicted data and
       test labels element by element
    accuracy = accuracy_score(labels_test,
       pred)
    return accuracy,pred

# Solving for Non Overlapping 4-Folds
   Cross Validation Accuracy
def getAccuracyfor4Folds(features, labels,
     classifier, iter=4):
    # Getting array indices
    index = np.arange(int(len(features)/4)
        , dtype=np.int)
    accuracy = 0

    # Getting accuracy for Non overlapping
        partitions
    for i in range(0,iter):
        # Making the index of test indices
            as true
        test_index = np.in1d(range(
           features.shape[0]), index)

        # Splitting the data
        features_train = features[~
           test_index]
        labels_train = labels[~test_index]
        features_test = features[
           test_index]
        labels_test = labels[test_index]

        # Getting the accuracy
        if classifier == 'LinearSVM':
            temp_accuracy,y_pred =
               classify_svm(
               features_train,
               labels_train,features_test
               ,labels_test)
        elif classifier == 'MLP':
            temp_accuracy,y_pred =
               classify_mlp(
               features_train,
               labels_train,features_test
               ,labels_test)
        elif classifier == 'DecisionTrees'
           :
            temp_accuracy,y_pred =
               classify_dc(features_train
               ,labels_train,
               features_test,labels_test)
        elif classifier == 'NaiveBayes':
            temp_accuracy,y_pred =
               classify_nb(features_train
               ,labels_train,
               features_test,labels_test)
        elif classifier == 'kNN':
            temp_accuracy,y_pred =
               classify_nn(features_train
               ,labels_train,
               features_test,labels_test)

        accuracy = accuracy +
           temp_accuracy

        # Incrementing the indices
        index = np.add(index,int(len(
           features)/4))

    return accuracy/iter

# Solving for Overlapping 4-Folds Cross
   Validation Accuracy and BER
def getAvgAccuracyandBER(features, labels,
     classifier, iter=10):
    avg_accuracy = 0
    BER = 0

    # Getting accuracy for Non overlapping
        partitions
    for i in range(0,iter):
        # Getting the unique indices
        # Sometimes a classes gets missed
           resulting in an all zero row
           of confusion matrix
        # Catered that in calculation BER
        index = np.random.choice(range(len
           (labels)), int(len(features)
           /4), replace=False)
        test_index = np.in1d(range(
           features.shape[0]), index)

        # Splitting the data
        features_train = features[~
           test_index]
        labels_train = labels[~test_index]
        features_test = features[
           test_index]
        labels_test = labels[test_index]

        # print "Total features for
           Training: " + str(len(
           features_train))
        # print "Total features for
           Training: " + str(len(
           features_test))

        # Getting the accuracy
        if classifier == 'LinearSVM':
            temp_accuracy,y_pred =
               classify_svm(
               features_train,
               labels_train,features_test
               ,labels_test)
        elif classifier == 'MLP':
            temp_accuracy,y_pred =
                classify_mlp(
                features_train,
                labels_train,features_test
                ,labels_test)
        elif classifier == 'DecisionTrees'
            :
            temp_accuracy,y_pred =
                classify_dc(features_train
                ,labels_train,
                features_test,labels_test)
        elif classifier == 'NaiveBayes':
            temp_accuracy,y_pred =
                classify_nb(features_train
                ,labels_train,
                features_test,labels_test)
        elif classifier == 'kNN':
            temp_accuracy,y_pred =
                classify_nn(features_train
                ,labels_train,
                features_test,labels_test)

        avg_accuracy = avg_accuracy +
            temp_accuracy

        #Getting BER
        BER = BER + get_BER(y_pred,
            labels_test)


    return BER/iter, avg_accuracy/iter


# Loads data from a .mat file
f = sio.loadmat('ProjectData.mat')

# Extracts specific column from data
hi_curve = f['HiCurve']
lo_curve = f['LoCurve']
labels = f['ClassLabels']

## Data Pre-processing
## Converts columns into feature vectors
features_hicurve = []
for elem in hi_curve:
    for item in elem:
        for data in item:
            # ignores some data points at
                end for consistency
            features_hicurve.append(data
                [:5500])
    #print(elem[0][0])

features_locurve = []
for elem in lo_curve:
    for item in elem:
        for data in item:
            features_locurve.append(data
                [:5500])


print("Length of Features(hicurve): " ,
    str(len(features_hicurve)))
print("Length of Features(locurve): " ,
    str(len(features_locurve)))

# Converting lists into numoy arrays
features_hi = np.array(features_hicurve)
features_lo = np.array(features_locurve)

# Concatenates hicurve and locurve into
    one feature vector
features = np.concatenate((
    features_hicurve,features_locurve),
    axis = 1)
print("Size of Concatetinated Features: "
    + str(features.shape))
#features = np.concatenate((hi_curve,
    lo_curve), axis=1)
# print(features.shape)


# Support Vector Machines
print("\n\n****Predicted Data (Support
    Vector Machine)*****")

accuracy = getAccuracyfor4Folds(features,
    labels, classifier = 'LinearSVM')
BER, avg_accuracy = getAvgAccuracyandBER(
    features, labels, classifier = '
    LinearSVM')
print("\nAverage Accuracy over 4 Folds(Non
     Overlapping Partitions): " + str(
    round(accuracy*100.00,2)) + "%")
print("Average Accuracy over 4 Folds(With
    Overlapping Partitions): " + str(round
    (avg_accuracy*100.0,2)) + "%")
print("BER: ", str(round(BER,3)))

# Decision Tree
print("\n\n****Predicted Data (Decision
    Tree)*****")

accuracy = getAccuracyfor4Folds(features,
    labels, classifier = 'DecisionTrees')
BER, avg_accuracy = getAvgAccuracyandBER(
    features, labels, classifier = '
    DecisionTrees')
print("\nAverage Accuracy over 4 Folds(Non
     Overlapping Partitions): " + str(
    round(accuracy*100.00,2)) + "%")
print("Average Accuracy over 4 Folds(With
    Overlapping Partitions): " + str(round
    (avg_accuracy*100.0,2)) + "%")
print("BER: ", str(round(BER,3)))
```

```
# Multilayer Perceptron
print("\n\n****Predicted Data (Multi Layer
     Neural Net)*****")

accuracy = getAccuracyfor4Folds(features,
    labels, classifier = 'MLP')
BER, avg_accuracy = getAvgAccuracyandBER(
    features, labels, classifier = 'MLP')
print("\nAverage Accuracy over 4 Folds(Non
     Overlapping Partitions): " + str(
    round(accuracy*100.00,2)) + "%")
print("Average Accuracy over 4 Folds(With
    Overlapping Partitions): " + str(round
    (avg_accuracy*100.0,2)) + "%")
print("BER: ", str(round(BER,3)))


# K- nearest Neighbours
print("\n\n****Predicted Data (k-Nearest
    Neighbour)*****")

accuracy = getAccuracyfor4Folds(features,
    labels, classifier = 'kNN')
BER, avg_accuracy = getAvgAccuracyandBER(
    features, labels, classifier = 'kNN')
print("\nAverage Accuracy over 4 Folds(Non
     Overlapping Partitions): " + str(
    round(accuracy*100.00,2)) + "%")
print("Average Accuracy over 4 Folds(With
    Overlapping Partitions): " + str(round
    (avg_accuracy*100.0,2)) + "%")
print("BER: ", str(round(BER,3)))


# Guassian naive Bayes
print("\n\n****Predicted Data (Guassian
    Naive Bayes)*****")

accuracy = getAccuracyfor4Folds(features,
    labels, classifier = 'NaiveBayes')
BER, avg_accuracy = getAvgAccuracyandBER(
    features, labels, classifier = '
    NaiveBayes')
print("\nAverage Accuracy over 4 Folds(Non
     Overlapping Partitions): " + str(
    round(accuracy*100.00,2)) + "%")
print("Average Accuracy over 4 Folds(With
    Overlapping Partitions): " + str(round
    (avg_accuracy*100.0,2)) + "%")
print("BER: ", str(round(BER,3)))
```

References

[1] Images Courtesy: https://www.google.com.pk

[2] kNN Classifier: https://www.google.com.pk

[3] Decision Trees: https://en.wikipedia.org/wiki/Decision_tree

[4] ML Libraries: http://scikit-learn.org

[5] Naive Bayes Classifier: http://scikit-learn.org/stable/modules/naive_bayes.html

[6] MLP Classifier: https://en.wikipedia.org/wiki/Multilayer_perceptron