vWrite

Writing in the Air

Human Computer Interaction on the Cloud

Saman Shafigh <saman.shafigh@gmail.com>

PhD Student - Computer Science and Engineering

COMP9417 Machine Learning and Data Mining
Final Project

University of New South Wales

Table of Contents

1 Introduction

Recent advances in Internet of things and wearable sensing devices capable of gathering and transmitting kinematic data from human body directly to Internet have provided us with the feasibility of implementing online human activity recognition systems. The combination of these new wearable sensing devices on Internet is viewed as a great potential [1], especially if the provided platforms are designed to be flexible and easy to use by developers. However, a typical wearable device is extremely resource constrained [2]. For example, the mica family only includes 8-bit processors with 4 Kbytes of data or program memory and 128 Kbytes of memory. This makes the implementation of hand gesture inference and classification algorithms impossible on such tiny devices especially for lazy learning algorithms like K-NN. Therefore, in this project (Fig.1) a web based API learning and classification platform is introduced from which received hand's gestures and particularly hand's writing related features are deduced and used to classify the user hand's movement in the air to the corresponding alphabet letters over the cloud. This cloud based API platform can be used by developers to leverage the use of kinematic wearable sensing devices in their game, communication, or health care applications.

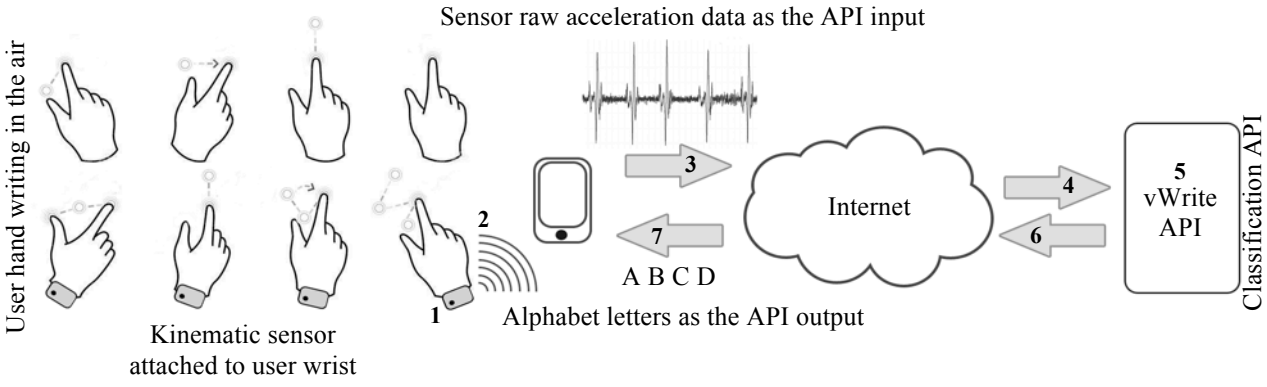


Fig.1 Accelerometer sensor attached to the user wrist (1) captures user hands movement and send raw data to user mobile phone (2) mobile phone then will send these data to the classification API on the cloud (3). vWrite receives sensor raw data (4) and will classify it to the corresponding letters (5). The letters will send it back to the client (6) to be used for other applications (7).

2 Definition of Learning Problem

The definition of machine learning problem in this work is to define and implement a system, which I call it 'vWrite' (Void Write) in the context of this work. It maps user's hand movements in the air to its corresponding alphabet letter and improves its knowledge and performance as it gains more experience. The key issues that have been addressed in this work regarding to the implementation of such system are explained in the following sub sections.

2.1 Type of algorithms for learning general target functions

The challenge that I had with my training data was related to the inconsistency in the dimension size of each training instance even when they belong to the same class; this issue has been explained in detail in Section 4.4. Considering the characteristics of my training data, I used Dynamic Time Warping (DTW) to measure the similarity between two temporal sequences, which may vary in time or speed and will then apply the K-NN algorithm to classify unseen instances.

2.2 Number of training data that is sufficient

Since a lazy learning algorithm (K-NN) is used in this work, the number of training instances can have a great impact on the performance and accuracy of the system while users work with it. In this work 234 training instances for 26 classes (A-Z letters) is used to achieve 97% of accuracy. Having 234 instances the system performs quite fast (579 ms) to find the K nearest neighbors. The effect that the number of training instances leaves on the execution time and the accuracy of the vWrite K-NN has been explained in detail in section 6.6 and 6.7.

2.3 The strategies for choosing a useful next training experience

The vWrit has some default training experiences in its database that I refer to them as Template Training Experiences (TTE). Beside the TTE I also ask users, prior to using the system for the first time, to write some predefined words and sentences to obtain new training experiences. I refer to these new training data as User Training Experiences (UTE).

2.4 How to improve the ability to represent and learn the target function

vWrite system can learn automatically, unobtrusively, and continuously over time to improve its ability to represent and learn the target function which is more specific to each user. While user works with this system I obtain new training data and add them to the system's database. I also use 3 bias rules to improve the ability of the system to represent and learn the target function that is more specific to each user. The first and basic bias rule is the distance based rule which I apply more penalty weight to those instances further to the query point (unseen instance) in my K-NN algorithm. The second bias rule is to apply more weight and credit to UTE instances over TTE instances. The last bias rule is to give more credit to those instances in database that have been selected more often. The implementation of these functions and accuracy of the system using them will be discussed in more details in Sections 5.3 and 6.4.

2.5 Summary of the vWrite learning problem

The learning problem in this work can be summarized as follows:

1. Type of Training Experience: Direct and supervised training instance obtained from kinematic sensor attached to user wrist while he/she writ alphabet letter in the air

2. Target Function (V): Map kinematic sensor data from user hand movement in the air to corresponding alphabet letter: MappToLetter: Kinematic Data → Letter

3. Representation of the Target Function: k-Nearest Neighbors algorithm with Dynamic Time Warping distance measurement is used to represent the target function V

4. Learning Algorithm: Several weight update rule learning algorithms are represented in this work to bias training instances

3 API Implementation

Node.js is used for implementation of my API with its following external packages (yamljs, mathjs, plotter, fs, fast-csv, dtw). I used Node.js because of its event-driven, non-blocking I/O model that makes it lightweight and efficient and more importantly its powerful Addons feature that can allow me to implement some computational-intensive parts of my system in C++.

Fig.2 illustrates the performance of Addons feature. I used a simple non optimized nth Fibonacci function implemented in C++ and Javascript for this test on a PC with the following specifications: Memory: 7.7 GiB, Processor: Intel® Core™ i7-3770, CPU: @3.40GHz × 8.

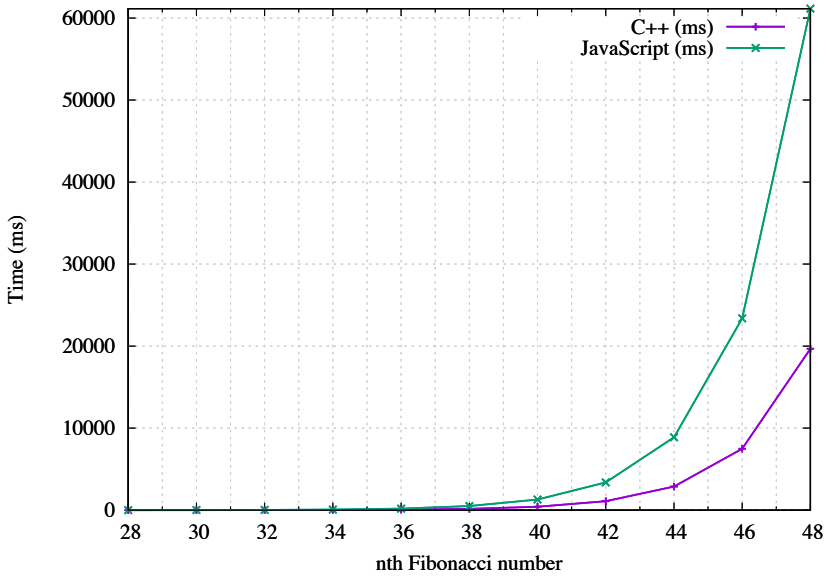


Fig.2 Time consumed for calculating the 28th to 48th Fibonacci number for an Addons C++ function compared to a Javascript function

4 Collection of Training Data

4.1 Raw Data Collection

To obtain data, a Shimmer 9DOF (9 degree of freedom) wireless sensor [3] shown in Fig. 3 has been used which combines the features of a 3-axis MEMS accelerometer, a 3-axis MEMS gyroscope and a 3-axis MEMS magnetometer to provide a kinematic wireless sensing solution.

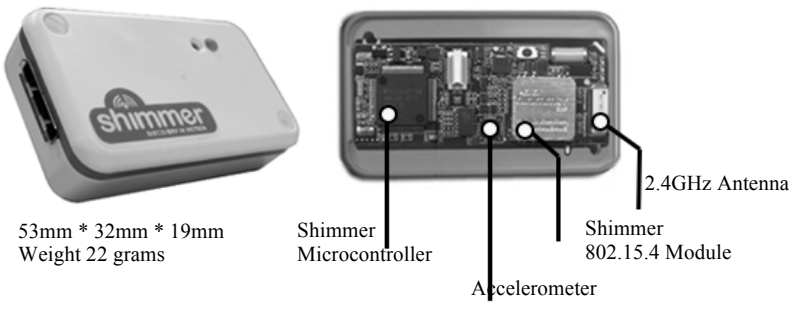


Fig. 3 Shimmer Wireless Sensor Unit

The Shimmer 9DOF wireless sensor is placed on a user's wrist (Fig.4) and transmits its data through the ZigBee [4] protocol to the base station (base station could be a user's smart phone or even a direct Internet connection if the sensor supports necessary protocols). A simple Python program is used to receive data on the USB port and saves them in a .csv file as the raw training data.

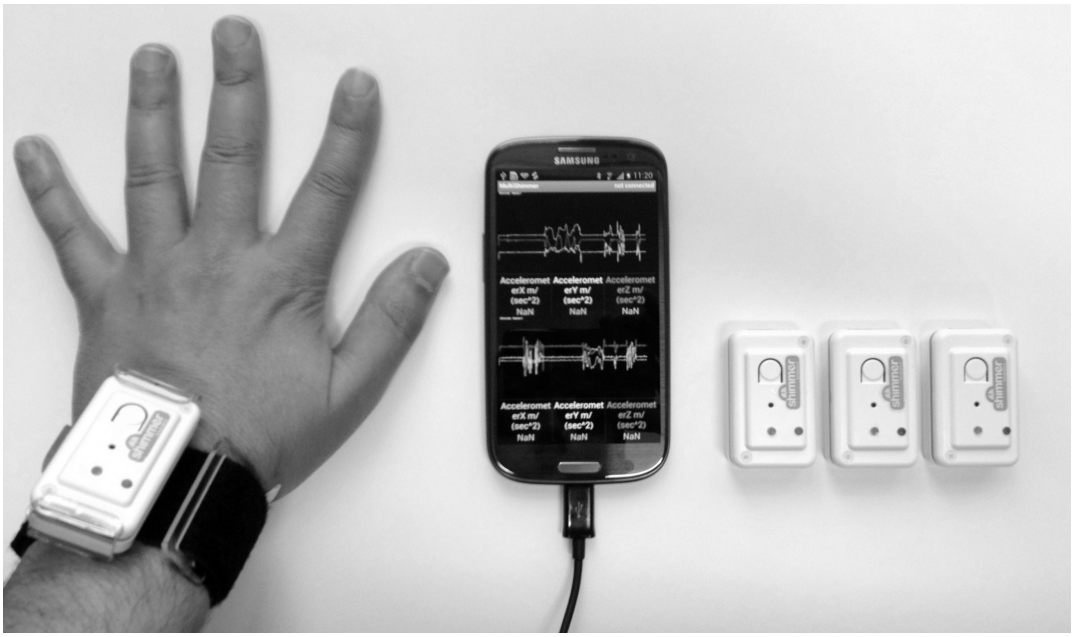


Fig.4 Shimmer Wireless Sensor attached to my wrist

9 raw training data were obtained for each letter (A-Z) or 234 training instances in total for all letters. Each training instance is obtained with sample rate of 100Hz and while the user tries to write each letter with a height of approximately 10cm in the air.

4.2 Data Calibration

The obtained raw sensors data must be calibrated before it can be used as training or unseen data otherwise the learning system will not have accurate and consistent behavior among other sensors since even two sensors from the same manufacturer may yield slightly different readings. For this purpose I used the accelerometer calibration technique explained in [5]. The calibrated sensor data is achieved through formula (1) and the configurations in (2). The value of these configurations can be obtained by Shimmer 9DOF calibration software [6].

$$c = R^{-1}.K^{-1}.(u - b) \qquad (1)$$

In formula (1) the alignment matrix *R* can be used to allow the user to define which axes is the x-axis, y-axis and z-axis and to define which direction is the positive direction. The sensitivity matrix *K* defines the sensitivity of each axis of the sensor. The offset vector *b* defines the zero offset for each axes of the tri-axial sensor.

$$c\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \text{Calibrated data}$$

$$R\begin{bmatrix} -1 & 0 & 0.01 \\ -0.01 & -1 & 0.01 \\ 0 & -0.04 & 1 \end{bmatrix} = \text{Alignment matrix}$$

$$K\begin{bmatrix} 96 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 98 \end{bmatrix} = \text{Sensitivity matrix}$$

$$u\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} = \text{Raw data}$$

$$b\begin{bmatrix} 1926 \\ 2150 \\ 1695 \end{bmatrix} = \text{Offset vector}$$

(2)

4.3 Feature Extraction

The Shimmer kinematic sensor as mentioned before provides nine features that can be used in a classification algorithm. Three features related to the magnetometer sensor are practically unusable due to the fact that they are related to user's orientation, which in my case is not consistent, and we will not mention them in this work. The remaining candidates that can be used are the three features for accelerometer and three features for gyroscope sensors. However in previous works such as [7, 8] it was shown that three acceleration features are the best candidate for activity recognition as also concluded in this work in Section 6.2. To fully justify my feature selections, I did a test on each feature (3D accelerometer and 3D gyroscope) and also the combination of them to compare the information gain of each feature and also the accuracy of the final target function, which will be explained in the result section.

4.4 Generating Training Experience

The training data in this work is generated in a supervised manner. Each training set is recorded in a separate .csv file named with the letter user intended to write in the air (e.g. A.csv, B.csv). Each file includes raw values of 6 dimension sensors data (3D accelerometer and 3D gyroscope) recorded over time for 10 training sample with an interval of 1 second between each sample. Figures 5 and 6 represent the change in the value of acceleration on the y-axis over time for training instances of letters A and B. In Appendix 2 some sample of classes A-Z also is provided.

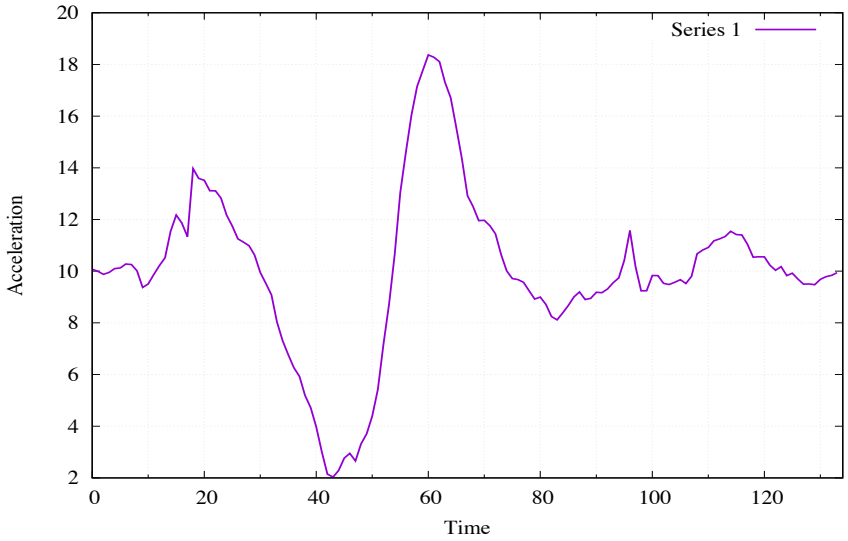


Fig. 5 change in the value of acceleration y-axis for a letter A

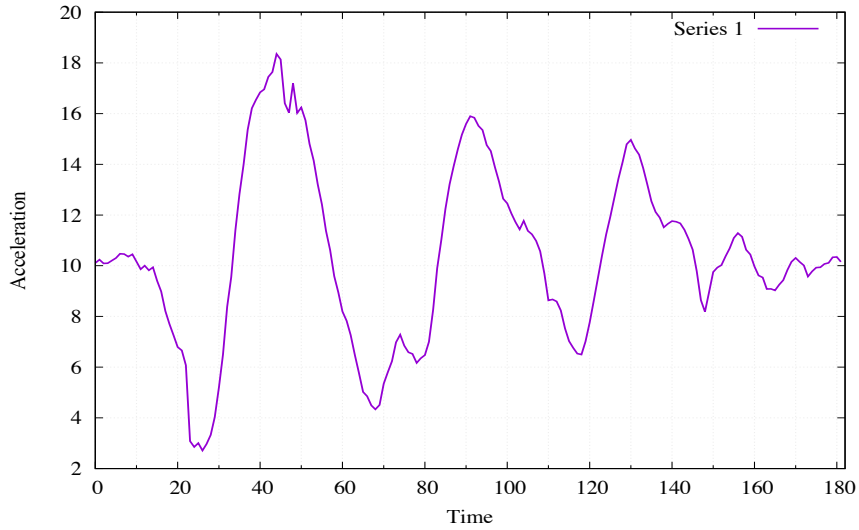


Fig. 6 change in the value of acceleration y-axis for a letter B

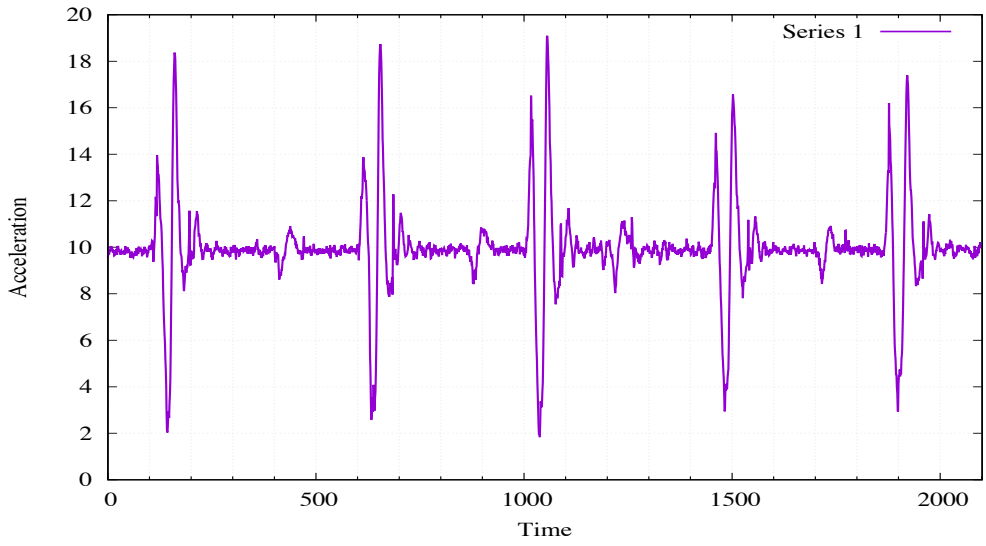


Fig. 7 change in the value of acceleration y-axis for 4 letters A

However each training sample is not just represented with six dimensions of data but also the change in the value of these six dimensions over the time it took the user to finish the writing of a particular letter in air. For example letter A in average is a matrix of 6*138 dimensions of data. To summaries a training sample is represented by a vector matrix of size N*M where size N is related to the 3D accelerometer and 3D gyroscope and size M is related to time.

Dimension of one training instance $\overset{\text{N}}{\overset{\text{M}}{\begin{bmatrix} a_{x1} & \cdots & a_{xi} \\ a_{y1} & & a_{yi} \\ a_{z1} & \ddots & a_{zi} \\ g_{x1} & & g_{xi} \\ g_{y1} & & g_{yi} \\ g_{z1} & \cdots & g_{zi} \end{bmatrix}}}$

Note that the size of M is strongly related to the sensor's sample rate value. The higher the sample rate the bigger the M and also higher resolution achieved for each letter, but on the other hand more data needs to be transmitted. The best sample rate value would be the smallest possible that can be achieved without compromising the accuracy of the clarification algorithm.

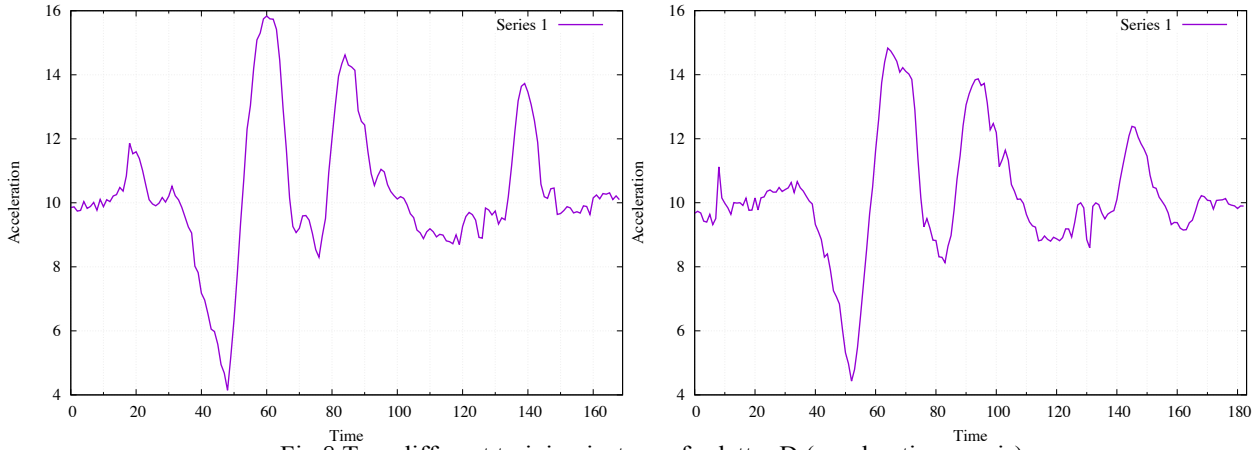


Fig.8 Tow different training instance for letter D (acceleration y-axis)

By a closer look at each training instance for example in Fig. 8 we can see the size of M is not consistent (170 in left and 185 in right) even among the training instances of the same target class. This is due to the fact that the speed of the user's hand writing in the air is not consistent even for a same letter. Max, min and average of M for each class A-Z is illustrated in Fig.9. This is the main reason I used dynamic time warping (DTW) algorithm which is suitable to measure the similarity between two temporal sequences that may vary over time or in speed. One example is to detect the similarities of dancing patterns even if one person dances faster than the other [9]

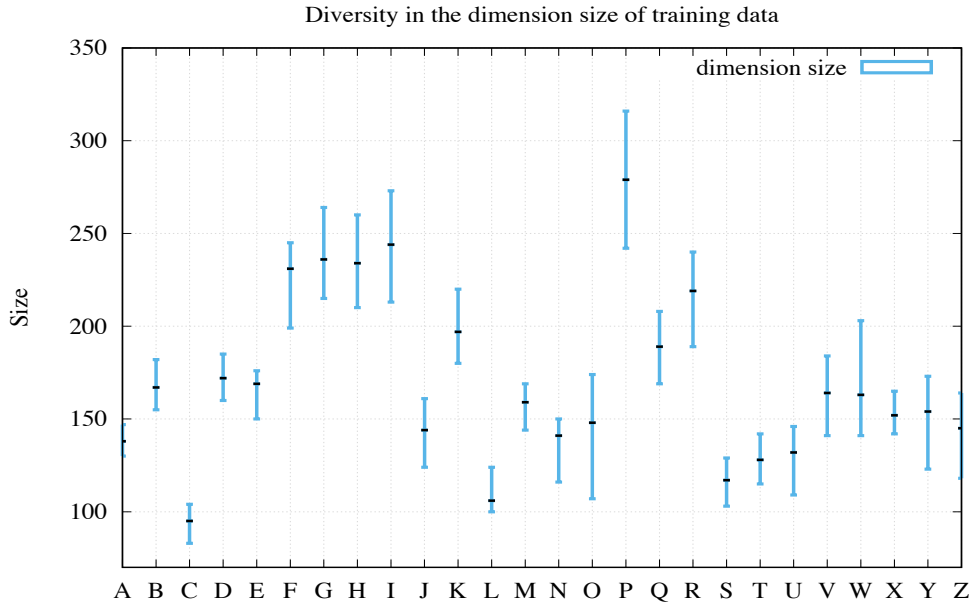


Fig.9 Diversity in the size of M for letters A-Z

5 Implementation of learning algorithm

5.1 Distance measurement

A JavaScript implementation [10] of DTW (Dynamic Time Warping) is used in vWrite to compute the similarity between two instances o1 [N*M1] and o2 [N*M2] where N represents the sensor available features and M1 and M2 representing the size of each instances over time. This function is then used to generate a distance vector of instance o1 from o2 based on the features that are defined in the vWrite configuration file (parameter classifier.dtw.distanceVector) to be used for generating the distance vector. A simple Euclidean function will then generate the value of distance given the distance vector and the features in the configuration file (parameter classifier.dtw.distanceVector) that are defined for the classification (obviously a feature used in classification must also be available in distance vector). The pseudo code of this implementation is as follows and the source code is avaliable in Appendix 1.

```
function getDistnace(o1, o2)
    set distance = 0
    vector distanceVector = getDistanceVector(o1, o2)
    for feature n in all features used for classification do
        distance = distance + pow(distanceVector[n], 2);
    return sqrt(sum)

function getDistanceVector(o1, o2)
    set distance = []
    for feature n in all features used for generating distance vector do
        distance[n] = dtw.compute(o1[n], o2[n])
    return distance;
```

The distance of each training instance compared to all training instances is calculated and illustrated. This data is available on the Github repository [11] of this project in tree/master/plot for each 234 training instances. The features that are used to generate the distance vector of two instances are

$$\left[\sqrt[2]{a_x^2 + a_y^2 + a_z^2} \quad \sqrt[2]{g_x^2 + g_y^2 + g_z^2}\right].$$

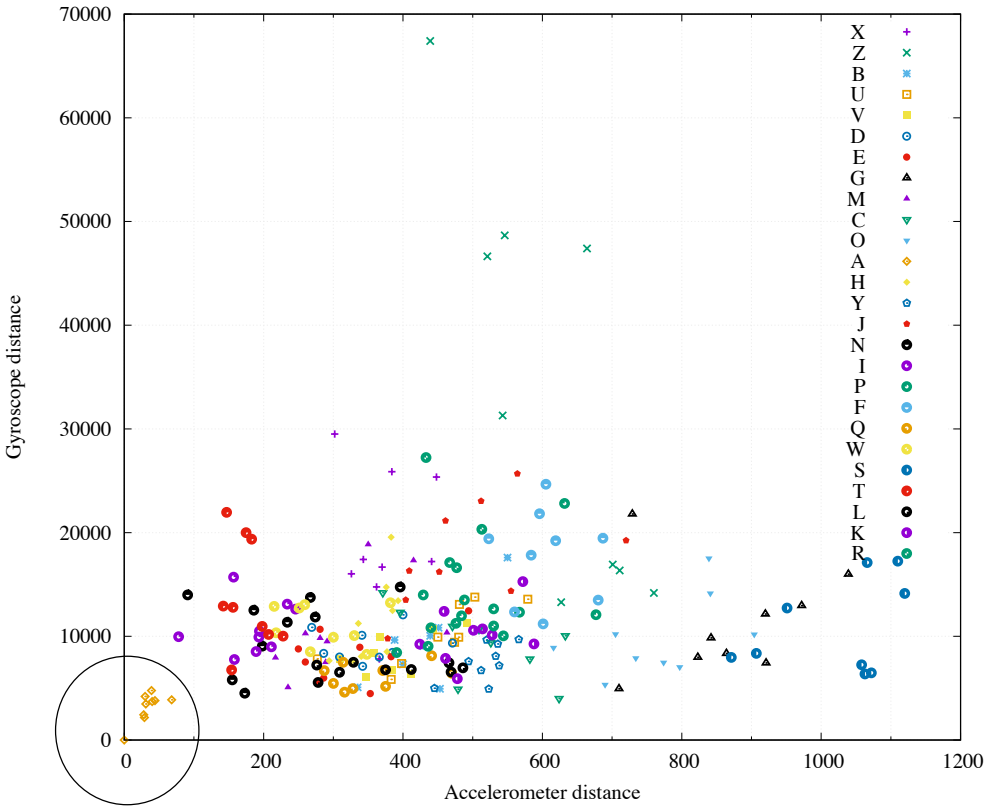


Fig.10 Distance of one training instance A from entire training instances

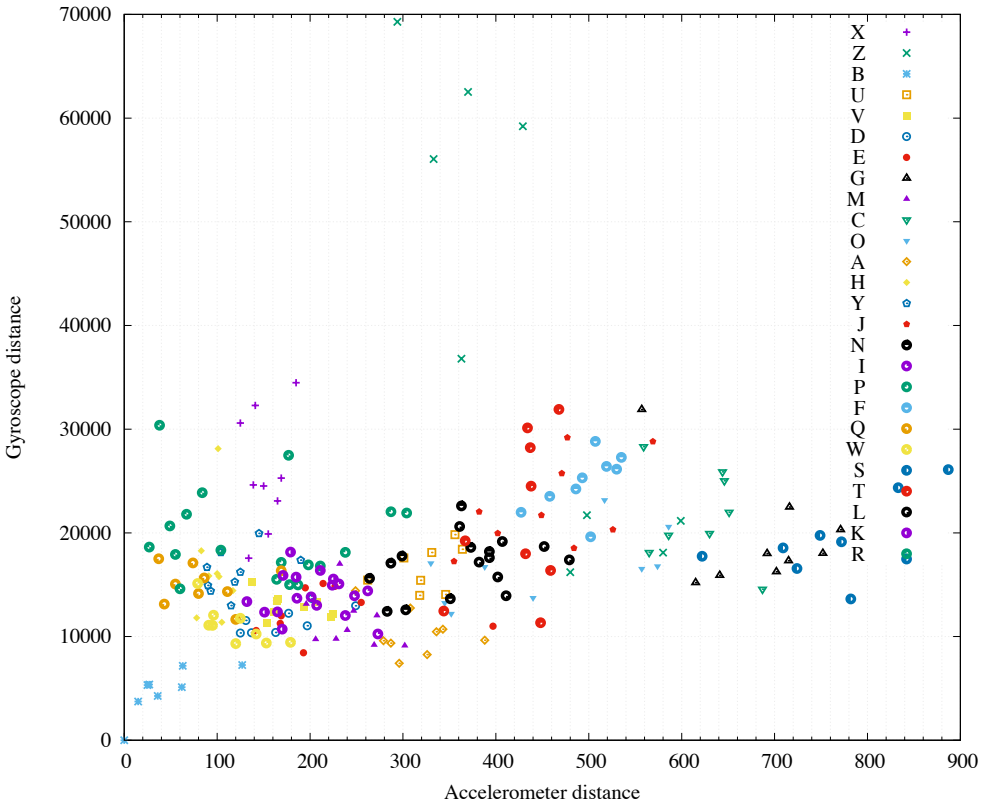


Fig.11 Distance of one training instance B from entire training instances

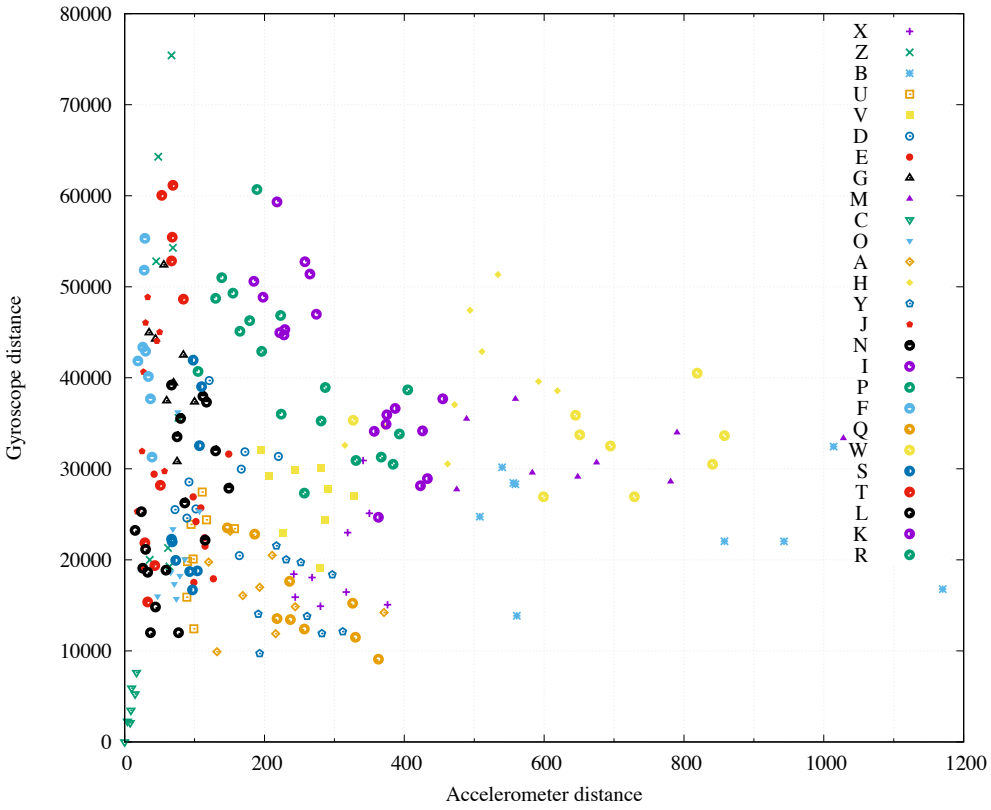


Fig.12 Distance of one training instance C from entire training instances

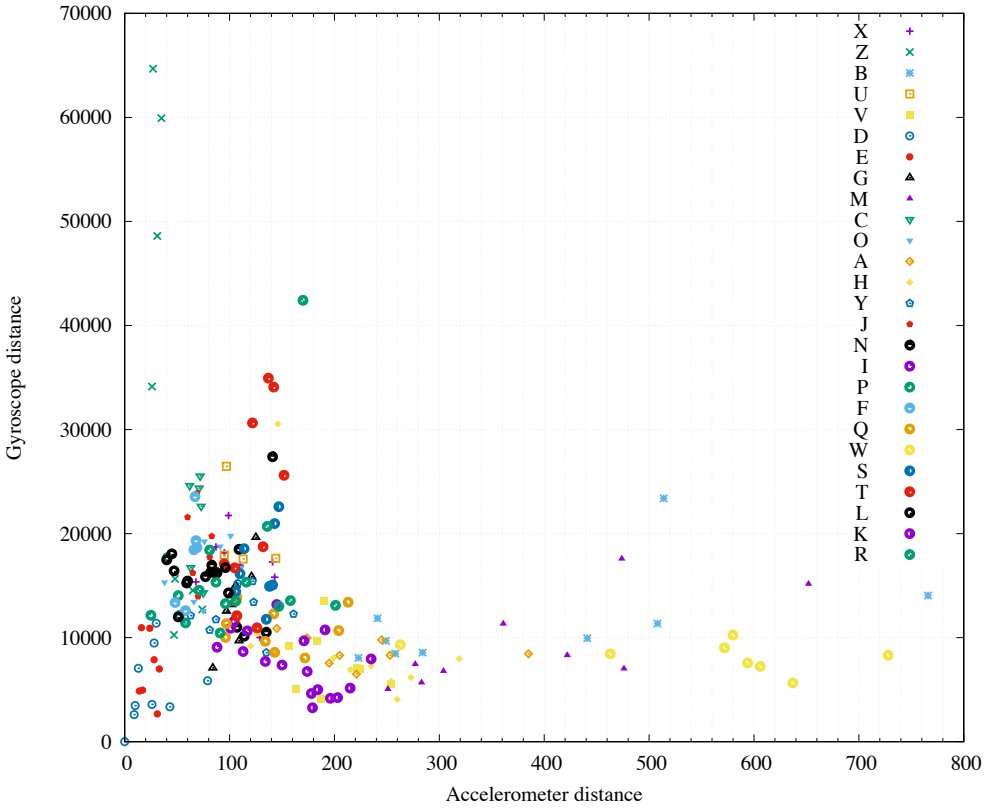


Fig.13 Distance of one training instance D from entire training instances

Fig.10, 11, 12, and 13 show the distances of one arbitrary query instance A, B, C and D from all training instances separately. Note that these graphs show only the relative distance of a query instance from all the training instances and they are not representing the distance of each training instance to another or how they are separated from each other. Since every instance is a N*M dimensional point in the space it is impossible to represent them on a 3D word.

5.2 Finding the k-Nearest Neighbors algorithm

The current version of vWrite uses a non-optimized way of finding the K nearest neighbors by simply comparing the distance of the query point with all training instances and then sorting the result and finally returning the first K instance in the list. The pseudo code of this implementation is as follows and the source code is available in Appendix 1.

```
function getKNN(query)
    set result = [], knn = []
    for training instance t in all training instances do
        add to result object ->
                  {'alias': t.alias, 'distance': getDistnace(query, t.data) }

    sort result based on distance field
    knn = get Kth from the beginning of result

    return knn
```

5.3 Voting between neighbors and the bias methods

Four voting methods are implemented in this work as below. The most basic one is a simple voting among the K nearest neighbors (candidate).

```
function simpleVote(candidates)
    set classWeight = {}, max = 0, electedClass = null
    for candidate c in candidates do
        if classWeight[c.alias] is defined
           classWeight[c.alias]++;
        else
           classWeight[c.alias] = 0

        // Select the class of max weight
        if frequency[c.alias] > max
           max = classWeight[c.alias]
           electedCandidate = c.alias

    return electedClass
```

The second method is to apply a distance penalty based on formula (3) for each neighbors.

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

Where $$w_i = \frac{1}{d(x_q, x_i)^2} \quad (3)$$

```
function distanceBasedVote(candidates)
    set classWeight = {}, max = 0, electedClass = null
    for candidate c in candidates do
        if classWeight[c.alias] is defined
           classWeight[c.alias] = classWeight[c.alias] + (1/pow(c.distance, 2))
        else
           classWeight[c.alias] = 0

        // Select the class of max weight
        if classWeight[c.alias] > max
           max = classWeight[c.alias]
           electedClass = c.alias

    return electedClass
```

The third method is to apply a bias on user's training instances beside the bias for the distance. This method simply adds an extra weight to UTE instances compared to TTE instances.

```
function userBiasBasedVote(candidates)
    set frequency = {}, max = 0, electedClass = null
    for candidate c in candidates do
        if classWeight[c.alias] is defined
           classWeight[c.alias] = classWeight[c.alias] +
                 (1/pow(c.distance, 2)) +
                 (c.isTraining * uteBias)
        else
           classWeight[c.alias] = 0

        // Select the class of max weight
        if classWeight[c.alias] > max
           max = classWeight[c.alias]
           electedClass = c.alias

    return electedClass
```

In this method the value of c.isTraining is 0 for TTE instances and 1 for UTE instances. uteBias is the weight of how important UTE instances could be.

The last method is to apply a bias on reputation of training instances beside the bias for the distance. The reputation of training instance indicates the number of times that training instance was previously selected as the elected candidate. The weight of reputation will then be calculated and added to the `class weight "classWeight"`.

```
function reputationBasedVote(candidates)
    set classWeight = {}, max = 0, electedClass = null, electedCandidate = null
    for candidate c in candidates do
        if classWeight[c.alias] is defined
           classWeight[c.alias] = classWeight[c.alias] +
               (1/pow(c.distance, 2)) +
               (c.reputation/numberOfInquiries)
        else
           classWeight[c.alias] = 0

        // Select the class of max weight
        if classWeight[c.alias] > max
           max = classWeight[c.alias]
           electedCandidate = c

    // Increase the reputation of elected candidate
    electedCandidate.reputation++;

    return electedCandidate.alias
```

6 Result

6.1 Validation technique

The cross validation technique is used to assess how well the target function performs in the classification of unseen instances and is referred to as accuracy it in this text. The way this cross validation is implemented is to remove one training instance at a time and use it as a query or unseen instance to find out how it is well classified by vWrite (the result of classification will be compared to the actual label of query instance). After that the query instance will be added again to the training database and we will continue on this action for the entire training instances. The cross validation value at the end will be calculated based on the total number of correct classifications versus the total number of correct and incorrect classifications. The pseudo code of this implementation is as follows and the source code is available in Appendix 1.

```
function crossValidation(T, voter)
    set correct = 0, incorrect = 0
    for training instance t in all training instances T do
        remove t from T
        candidates = getKNN(t, T)
        elected = voter(candidates)
        if elected.alias == t.alias
           correct++
        else
           incorrect++;
        add t to T

    return (correct/correct + incorrect) * 100
```

In this code the injected voter could be any of voting methods mentioned in section 5.3.

6.2 Accuracy and feature selection

Fig.14 shows the use of different features and also the combination of these features in the vWrite classification algorithm and their impact on the accuracy. As mentioned in Section 4.3 we can see that acceleration features perform much better than gyroscope features. One concept that we can get from this figure is that the combination of acceleration features Y and Z gives us the most accuracy (97%) compared to other options. This is due to the fact that writing a letter in the air is mostly involved with the movement of user in just two directions, which in this case is Y and Z. The other concept we can get from this graph by looking at A-Z or A-Y is that even a sensor with only one dimension acceleration feature can effectively perform with an accuracy of 95%.

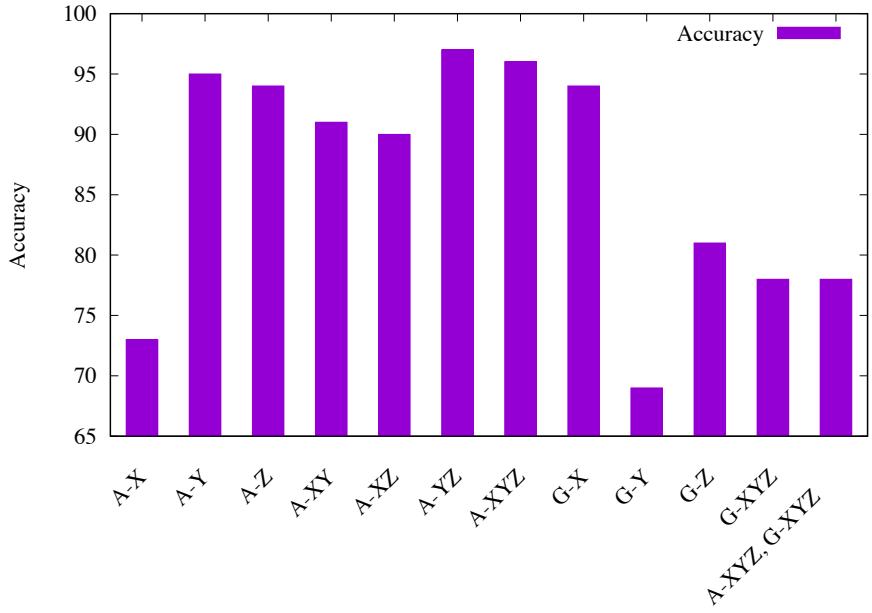


Fig.14 Accuracy and feature selection

6.3 Accuracy and number of neighbors (k)

Fig.15 shows the value of K in K-NN and its effects on the accuracy. Considering the fact that only 8 instances (1 instance is removed from 9 instances for cross validation) are provided in database for each 26 classes of A-Z we can therefore justify why the accuracy of simpleVote method drops when K passes the value of 6. However for distanceBasedVote the accuracy stays at 95% since we apply the distance penalty to all the selected neighbors.

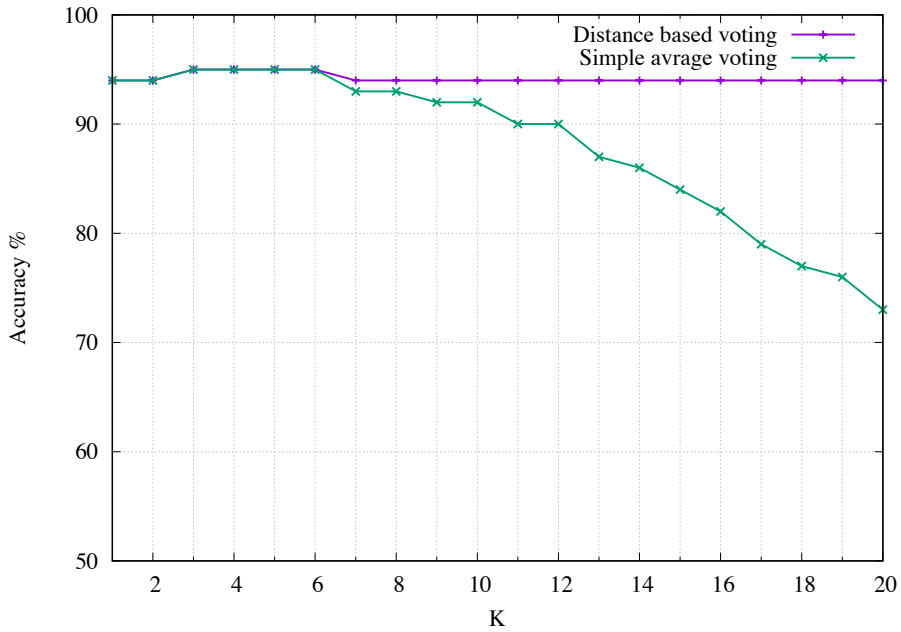


Fig.15 Accuracy and number of neighbors (k)

The following table shows the accuracy of vWrite with respect to each class A-Z for K=1 to K=10. From this table it is clear that for some letters like letter D and E the vWrite performs not quite well due to the similarity (Note the similarity is not how they look visually) of these letters Fig.13 and Appendix 2 (D, E). However for future work we can implement and use some other learning methods beside K-NN if an instance is calcified as one of these letters.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	100	100	100	67	78	89	100	100	100	89	100	100	100
2	100	100	100	67	78	89	100	100	100	89	100	100	100
3	100	100	100	44	89	89	100	100	100	89	100	100	100
4	100	100	100	56	78	89	100	100	100	89	100	100	100
5	100	100	100	56	89	89	100	100	100	89	100	100	100
6	100	100	100	44	89	89	100	100	100	89	100	100	100
7	100	100	100	44	89	89	100	100	100	89	100	100	100
8	100	100	100	33	89	100	100	100	100	89	100	100	100
9	100	100	100	33	89	100	100	100	100	89	100	100	100
10	100	100	100	33	89	89	100	100	100	89	100	100	100

	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	100	100	100	89	78	89	100	100	89	89	100	89	100
2	100	100	100	89	78	89	100	100	89	89	100	89	100
3	100	100	100	89	89	89	100	100	100	89	100	100	100
4	100	100	100	89	100	89	100	100	100	89	100	100	100
5	100	100	100	89	78	89	100	100	100	89	100	100	100
6	100	100	100	89	89	89	100	100	89	89	100	100	100
7	100	100	100	89	89	89	100	100	100	89	100	100	100
8	100	100	100	89	78	89	100	100	100	89	100	100	100
9	100	100	100	89	78	89	100	100	100	89	100	100	100
10	100	100	100	89	89	89	100	100	100	89	100	100	100

Table 1 The accuracy of vWrite in respect to each class A-Z for K=1 to K=10

6.4 Accuracy and number of training experiences

Fig.16 shows the accuracy of vWrite with respect to number of training experiences. As we increase the number of training instances the accuracy of system also increases.

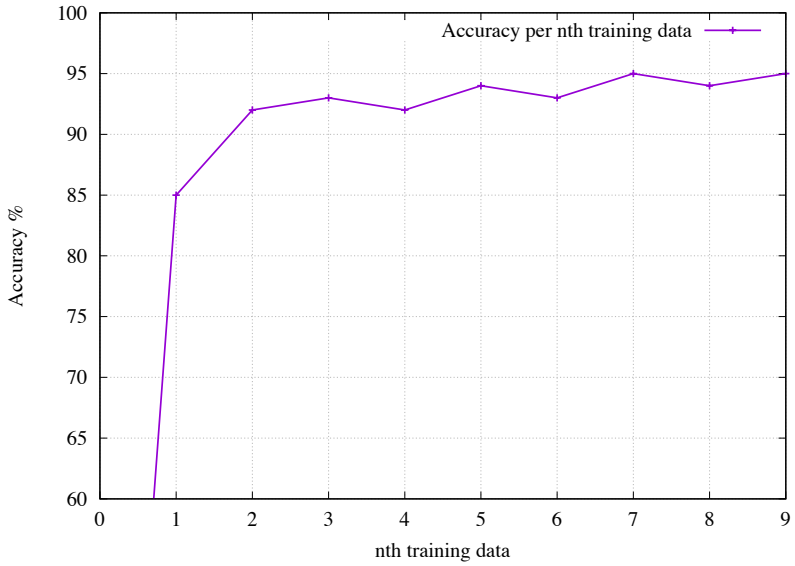


Fig.16 Accuracy and number of training experiences

6.5 Execution time and number of training experiences

The complexity of vWrite is O(n). Fig.17 shows the time in (ms) that is required for the K-NN in vWrite to select the K nearest neighbor of a query point. In this test I used my 234 original training instances and duplicate them each time (from 1 to 10) to perform this test. Note that although my data is duplicated but still the time that is required to run a simple K-NN search over these data is same as if the data is not duplicated.

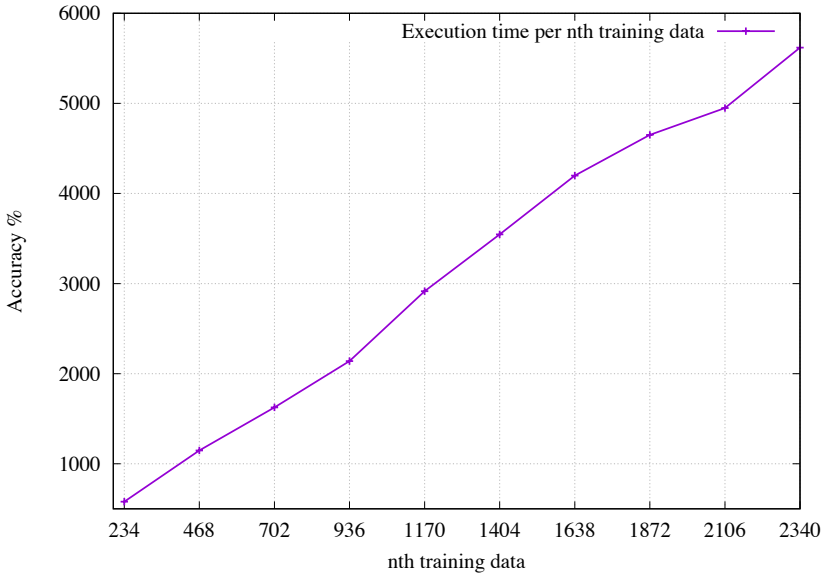


Fig.17 Execution time and number of training experiences

Appendix 1 (Source code)

```
vWrite
  |
  |-data
  |   |-raw
  |
  |-module
  |  |-classifier
  |  |    |-dtw.js
  |  |    |-knn.js
  |  |
  |  |-calibrator.js
  |  |-extractor.js
  |  |-plotter.js
  |  |-splitter.js
  |
  |-node_modules
  |
  |-app.js
  |-config.yml
```

```
app.js
--------------------------------------
var yaml =          require('yamljs');
var ploter =        require('./module/plotter');
var extractor =     require('./module/extractor');
var splitter =      require('./module/splitter');
var calibrator =    require('./module/calibrator');
var dtwClassifier = require('./module/classifier/dtw');
var knn =           require('./module/classifier/knn');
var math =          require('mathjs');

var config = yaml.load('config.yml');
var trainingClasses = config.training;
//trainingClasses = trainingClasses.slice(0, 2);

main();

function main() {
    var trainingData = [];
    splitter.getDimensionDiversity(trainingClasses);

    for (var key in trainingClasses) {
        if (trainingClasses.hasOwnProperty(key)) {
            // Preparing training data
            // Extracting and splitting training sets of each character
            var item = trainingClasses[key];
            extractor.extract(item, function(data, item) {
                // Extracting, calibrating and selecting training data for each character
                console.log("Extracting, calibrating and selecting " +
                            "training data for character: " + item.alias);
                var calibratedDate = calibrator.calibrate(data);

                // Plot some of training instances in one plot
                ploter.plotData(item, calibratedDate, item.boundary, 4);
                var instances = splitter.getInstances(calibratedDate, item.boundary);

                // Generate plots for each training instance
                console.log("Extracting and splitting character " + item.alias);
                ploter.plotTrainingData(item, instances);

                // Aggregating all training instances
```

```
                if (trainingData.length === trainingClasses.length) {
                    // Calculate the distance of each training instance from other
                    // training instances
                    console.log("Calculating distance matrix ...");
                    var distanceMatrix = dtwClassifier.getDistanceMatrix(trainingData);
                    ploter.plotDtwData(distanceMatrix);

                    // Performing cross validation for K = 1 to K = 20
                    console.log("Performing cross validation: ");
                    for (var K = 1; K < 20; K++) {
                        console.log("Calculate cross validation for K = " + K + " ...");
                        var accuracy = knn.crossValidation(
                            trainingData, // Training data
                            knn.simpleVote, // Voting method for the k nearest neighbors
                            K // Number of neighbors
                            );

                        console.log("Accuracy K[" + K + "]: " + accuracy);
                    }
                }
            });
        }
    }
}

function main2() {
    var trainingData = [];
    for (var key in trainingClasses) {
        if (trainingClasses.hasOwnProperty(key)) {
            // Preparing training data
            // Extracting and splitting training sets of each character
            var item = trainingClasses[key];
            extractor.extract(item, function(data, item) {
                var calibratedDate = calibrator.calibrate(data);
                // Plot some of training instances in one plot
                var instances = splitter.getInstances(calibratedDate, item.boundary);

                // Aggregating all training instances
                trainingData.push({'item': item, 'instances': instances});
                if (trainingData.length === trainingClasses.length) {
                    // Performing cross validation for K = 1 to K = 20
                    var K = 4;
                    var accuracy = knn.crossValidation(
                        trainingData, // Training data
                        knn.distanceBasedVote, // Voting method for the k nearest neighbors
                        K, // Number of neighbors
                        false // Debug mode
                        );

                    console.log(K + ": " + accuracy);
                }
            });
        }
    }
}
```

```
function main3() {
    var trainingData = [];
    var numberOfTrainingInstancesPerClass = 0;
    for (var key in trainingClasses) {
        if (trainingClasses.hasOwnProperty(key)) {
            // Preparing training data
            // Extracting and splitting training sets of each character
            var item = trainingClasses[key];
            extractor.extract(item, function(data, item) {
                var calibratedDate = calibrator.calibrate(data);
                // Plot some of training instances in one plot
                var instances = splitter.getInstances(calibratedDate, item.boundary);
                var nInstances = [];
                for (var i = 0; i < 10; i++) {
                    for (var j = 0; j < instances.length; j++) {
                        nInstances.push(instances[j]);
                    }
                }
                numberOfTrainingInstancesPerClass = nInstances.length;

                // Aggregating all training instances
                trainingData.push({'item': item, 'instances': nInstances});
                if (trainingData.length === trainingClasses.length) {
                    var subject = trainingData[0].instances.shift();

                    var start = new Date().getTime();
                    knn.getKNN(subject, trainingData);

                    var end = new Date().getTime();
                    var time = end - start;
                    console.log('Execution time to perform KNN on ' +
                            (numberOfTrainingInstancesPerClass * trainingClasses.length) +
                            ' instances: ' + time + ' milliseconds');
                }
            });
        }
    }
}
```

config.yml

```
-------------------------------------
classifier:
    dtw:
        distanceVector: [1, 8]
        distanceDimension: [0, 1]
    knn:
        K: 5
        uteBias: 0.5

calibration:
        -
            Ra: [[-1, 0, 0.01], [-0.01, -1, 0.01], [0, -0.04, 1]]
            Ka: [[96, 0, 0], [0, 100, 0], [0, 0, 98]]
            Ba: [[1926], [2150], [1695]]
            offset: [1, 2, 3]
        -
            Ra: [[-0.01, -1, 0.03], [-1, 0.01, -0.01], [0.02, -0.01, -1]]
            Ka: [[2.77, 0, 0], [0, 2.77, 0], [0, 0, 2.8]]
            Ba: [[1830], [1836], [1881]]
            offset: [4, 5, 6]
        -
            Ra: [[1, 0, 0], [0, 1, 0], [0, 0, -1]]
            Ka: [[706, 0, 0], [0, 708, 0], [0, 0, 629]]
            Ba: [[-9], [149], [90]]
            offset: [7, 8, 9]
```

```
training:
    -
      sample: ./data/raw/A.csv
      map: [0, 0, 0, 0, 0]
      alias: A
      boundary:
        - [2964, 3098]
        - [3462, 3598]
        - [3867, 4000]
        - [4309, 4439]
        - [4722, 4865]
        - [5084, 5225]
        - [5473, 5620]
        - [5840, 5984]
        - [6240, 6373]
    -
 ...
```

```
calibrator.js
--------------------------------------
var math = require('mathjs');
var yaml = require('yamljs');
var config = yaml.load('config.yml');

// Get calibration parameters and generate calibration matrix
var calibrationParams = config.calibration;
var calibrationMatrixs = [];
for (var j = 0; j < 3; j++) {
    var calParam = calibrationParams[j];
    calibrationMatrixs.push({
        'Ba': math.matrix(calParam.Ba),
        'iRa': math.inv(math.matrix(calParam.Ra)),
        'iKa': math.inv(math.matrix(calParam.Ka))
    });
}

// calibrate sensor data
exports.calibrate = function (data) {
    var calibratedData = [];
    for (var i = 0; i < data.length; i++) {
        var row = [];
        row.push(i); // 0
        for (var j = 0; j < 2; j++) {
            var calibrationMatrix = calibrationMatrixs[j];
            var calibrationParam = calibrationParams[j];
            var offset = calibrationParam.offset;

            var x = math.number(data[i][offset[0]]);
            var y = math.number(data[i][offset[1]]);
            var z = math.number(data[i][offset[2]]);

            // Sensor raw data
            var rawData = math.matrix([[x], [y], [z]]);
            // Sensor calibrated data
            var calData = math.multiply(
                    math.multiply(calibrationMatrix.iRa, calibrationMatrix.iKa),
                    math.subtract(rawData, calibrationMatrix.Ba)
                );

            var cx = math.round(calData.subset(math.index(0, 0)), 3);
            var cy = math.round(calData.subset(math.index(1, 0)), 3);
            var cz = math.round(calData.subset(math.index(2, 0)), 3);
            var cxyz = math.round(math.sqrt(math.pow(cx,2)+math.pow(cy,2)+math.pow(cz,2)),3);
```

```
            row.push(cxyz); // 1 + offset
            row.push(cx);   // 2 + offset
            row.push(cy);   // 3 + offset
            row.push(cz);   // 4 + offset
            row.push(x);    // 5 + offset
            row.push(y);    // 6 + offset
            row.push(z);    // 7 + offset
        }
        calibratedData.push(row);
    }

    return calibratedData;
};
```

extractor.js

```
var fs = require('fs');
var csv = require('fast-csv');

exports.extract = function (item, callback) {
    var stream = fs.createReadStream(item.sample);
    var dataBag = [];
    csv.fromStream(stream)
        .on("data", function(data) {
            if (!isNaN(data[1])) {
                dataBag.push(data);
            }
        })
        .on("end", function() {
            callback(dataBag, item);
        });
};
```

plotter.js

```
var plot = require('plotter').plot;
var math = require('mathjs');
var splitter =   require('./splitter');

// Plot each training instances in a seprate file
exports.plotTrainingData = function(item, instances, dimension) {
    if (dimension === undefined) {
        dimension = 1;
    }

    for (var i = 0; i < instances.length; i++) {
        var data = splitter.getOneDimension(instances[i].data, dimension);

        plot({
            data:       data,
            filename:   './plot/raw/' + item.alias + '-' + dimension + '-' + i + '.eps',
            options: [
                'grid xtics lt 0 lw 0 lc rgb "#eeeeee"',
                'xrange [0:' + data.length + ']',
                'xlabel "Time"',
                'ylabel "Acceleration"',
                'grid ytics lt 0 lw 0 lc rgb "#eeeeee"',
                'terminal postscript eps enhanced color font "Times Roman,18"'
            ]
        });
    }
};
```

```
// Plot all training instances of each class in one file
exports.plotData = function(item, data, boundary, numberOfSamples, dimension) {
    if (numberOfSamples === undefined) {
        numberOfSamples = boundary.length;
    }
    if (dimension === undefined) {
        dimension = 1;
    }

    var offset = 100;
    var start = boundary[0][0] - offset;
    var end = boundary[numberOfSamples][1] + offset;
    data = splitter.getOneDimension(data.slice(start, end), dimension);

    plot({
        data:       data,
        filename:   './plot/raw/' + item.alias + '-' + dimension + '.eps',
        options: [
            'grid xtics lt 0 lw 0 lc rgb "#eeeeee"',
            'xrange [0:' + data.length + ']',
            'xlabel "Time"',
            'ylabel "Acceleration"',
            'grid ytics lt 0 lw 0 lc rgb "#eeeeee"',
            'terminal postscript eps enhanced color font "Times Roman,18"'
        ]
    });
};

// Plot distance of one arbitrary instance form others.
exports.plotDtwData = function(distanceMatrix) {
    for (var ii = 0; ii < 9; ii++) {
        for (var i = 0; i < distanceMatrix.length; i++) {
            var from = distanceMatrix[i];
            var distanceTo = from.distances;
            var data = new Object();
            for (var j = 0; j < distanceTo.length; j++) {
                var to = distanceTo[j];
                data[to.alias] = new Object();
                console.log(from.alias + " -> " + to.alias + ":");
                var df = to.value[ii];
                for (var jj = 0; jj < df.length; jj++) {
                    var dt = df[jj];
                    data[to.alias][dt[0]] = dt[1];
                }
            }

            plot({
                data:       data,
                filename:   './plot/dtw/' + from.alias + '-' + ii + '.eps',
                style: 'points',
                options: [
                    'grid xtics lt 0 lw 0 lc rgb "#eeeeee"',
                    'grid ytics lt 0 lw 0 lc rgb "#eeeeee"',
                    'xlabel "Accelerometer distance"',
                    'ylabel "Gyroscope distance"',
                    'size 1.2,1.4',
                    'terminal postscript eps enhanced color font "Times Roman,18"'
                ]
            });
        }
    }
};
```

splitter.js

```
------------------------------------
var math = require('mathjs');
var frame = 10;
var recall = 10;
var thresholdValue = 200;
var thresholdFrame = 5;

exports.getInstances = function (dataBag, boundary, isTraining, length) {
    var instances = [];
    if (length === undefined) {
        length = boundary.length;
    }
    if (isTraining === undefined) {
        isTraining = true;
    }
    for (var i = 0; i < length; i++) {
        instances.push({
            'data': dataBag.slice(boundary[i][0], boundary[i][1]),
            'specs': {
                'isTraining': isTraining? 0 : 1,
                'reputation': 0
                }
            });
    }
    return instances;
};

exports.getOneDimension = function(dataBag, dimension) {
    var data = [];
    for (var i = 0; i < dataBag.length; i++) {
        data.push(dataBag[i][dimension]);
    }
    return data;
};

exports.getDimensionDiversity = function (trainingInfo) {
    var dimensionDiversity = [];
    for (var i = 0; i < trainingInfo.length; i++) {
        var set = trainingInfo[i];
        var max = null;
        var min = null;
        var sum = 0;
        for (var j = 0; j < set.boundary.length; j++) {
            var boundary = set.boundary[j];
            var size = boundary[1] - boundary[0];
            sum = sum + size;
            if (size > max || max === null) {
                max = size;
            }
            if (size < min || min === null) {
                min = size;
            }
        }
        console.log((i + 1) + '' + min + '' + math.round(sum/set.boundary.length) + '' + max);
        dimensionDiversity.push({
            'alias': set.alias,
            'min': min,
            'avre': math.round(sum/set.boundary.length),
            'max': max
        });
    }

    return dimensionDiversity;
};
```

```
exports.split = function (dataBag) {
    var data = [];
    var distanceData = [];
    var frameIndex = 0;
    var framesH = 0;
    var framesL = 0;

    var buffering = {
        start: true,
        end: true,
        startIndex: null,
        endIndex: null
    };

    var buffer = [];
    var bufferInfo = [];
    var splittedData = [];
    var length = dataBag.length;

    for (var i = 0; i < length; i+=recall) {
        if ((i + frame) < length) {
            frameIndex++;
            var sum = 0;
            for (var j = i; j < (i + frame); j++) {
                sum = sum + dataBag[j];
            }

            var mean = Math.round(sum / frame);
            var distance = 0;
            for (var j = i; j < (i + frame); j++) {
                distance = distance + Math.abs(dataBag[j] - mean);
            }

            distanceData.push(distance);
            if (thresholdValue < distance) {
                framesH++;
                if (thresholdFrame < framesH) {
                    buffering.end = true;
                    framesH = 0;
                    framesL = 0;
                    if (buffering.start === true) {
                        buffering.start = false;
                        buffering.startIndex = i;
                        buffer = [];
                    }
                    for (var j = i; j < (i + frame); j++) {
                        buffer.push(dataBag[j]);
                    }
                }
            } else {
                framesL++;
                if (thresholdFrame < framesL) {
                    if (buffering.end === true) {
                        buffering.end = false;
                        buffering.endIndex = i;
                        splittedData.push(buffer);
                        bufferInfo.push({
                            startIndex: buffering.startIndex,
                            endIndex: buffering.endIndex
                        });
                    }
                    buffering.start = true;
                    framesL = 0;
                    framesH = 0;
                }
```

```
            }
        }
    }

    return splittedData;
};


dtw.js
-------------------------------------
var yaml = require('yamljs');
var math = require('mathjs');
var DTW = require('dtw');
var splitter = require('../splitter');

var config = yaml.load('config.yml');
var classifier = config.classifier.dtw;
var dtw = new DTW();

function getDistanceVector(from, to) {
    var distance = [];
    for (var i = 0; i < classifier.distanceVector.length; i++) {
        var value = math.round(dtw.compute(
            splitter.getOneDimension(from, classifier.distanceVector[i]),
            splitter.getOneDimension(to, classifier.distanceVector[i])
            ));

        distance.push(value);
    }

    return distance;
}

function getEuclideanDistance(from, to) {
    var distanceVactor = getDistanceVector(from, to);
    var sum = 0;
    for (var i = 0; i < classifier.distanceDimension.length; i++) {
        var dimension = classifier.distanceDimension[i];
        sum = sum + math.pow(distanceVactor[dimension], 2);
    }

    return math.round(math.sqrt(sum));
}

function getDistanceMatrix(data) {
    var distanceMatrix = [];
    for (var i = 0; i < data.length; i++) {
        var from = data[i];
        var distanceTo = [];
        for (var j = 0; j < data.length; j++) {
            var to = data[j];
            var df = [];
            for (var ii = 0; ii < from.instances.length; ii++) {
                var dt = [];
                for (var jj = 0; jj < to.instances.length; jj++) {
                    dt[jj] =
                              getDistanceVector(
                                     from.instances[ii].data,
                                     to.instances[jj].data
                              );
                }
                df[ii] = dt;
            }
```

```
            distanceTo.push({'alias': to.item.alias, 'value': df});
        }
        distanceMatrix.push({'alias': from.item.alias, 'distances': distanceTo});
    }

    return distanceMatrix;
}

exports.getDistanceVector = getDistanceVector;
exports.getEuclideanDistance = getEuclideanDistance;
exports.getDistanceMatrix = getDistanceMatrix;
```

knn.js

```
var yaml = require('yamljs');
var math = require('mathjs');
var dtwClassifier = require('./dtw');
var config = yaml.load('config.yml');
var knnConfig = config.classifier.knn;
var numberOfInquiries = 0;

// Vote between candidate
function simpleVote(candidates) {
    var max = 0;
    var classWeight = {};
    var electedClass;

    for (var i = 0; i < candidates.length; i++) {
        var candidate = candidates[i];
        classWeight[candidate.alias] =
                (classWeight[candidate.alias] || 0) +  1;

        if (classWeight[candidate.alias] > max) {
            max = classWeight[candidate.alias];
            electedClass = candidate.alias;
        }
    }

    return electedClass;
}

// Distance penalty voting
function distanceBasedVote(candidates) {
    var max = 0;
    var classWeight = {};
    var electedClass;

    for (var i = 0; i < candidates.length; i++) {
        var candidate = candidates[i];
        classWeight[candidate.alias] =
                (classWeight[candidate.alias] || 0) +
                (1/math.pow(candidate.distance, 2));

        if (classWeight[candidate.alias] > max) {
            max = classWeight[candidate.alias];
            electedClass = candidate.alias;
        }
    }

    return electedClass;
}
```

```
// Distance penalty voting + user bias
function userBiasBasedVote(candidates) {
    var max = 0;
    var classWeight = {};
    var electedClass;

    for (var i = 0; i < candidates.length; i++) {
        var candidate = candidates[i];
        classWeight[candidate.alias] =
                (classWeight[candidate.alias] || 0) +
                (1/math.pow(candidate.distance, 2)) +
                (candidate.specs.isTraining * knnConfig.uteBias);

        if (classWeight[candidate.alias] > max) {
            max = classWeight[candidate.alias];
            electedClass = candidate.alias;
        }
    }

    return electedClass;
}

// Distance penalty voting + reputation bias
function reputationBasedVote(candidates) {
    var max = 0;
    var classWeight = {};
    var electedClass;
    for (var i = 0; i < candidates.length; i++) {
        var candidate = candidates[i];
        classWeight[candidate.alias] =
                (classWeight[candidate.alias] || 0) +
                (1/math.pow(candidate.distance, 2)) +
                (candidate.specs.reputation/numberOfInquiries);

        if (classWeight[candidate.alias] > max) {
            max = classWeight[candidate.alias];
            electedClass = candidate.alias;
        }
    }

    electedClass.specs.reputation = electedClass.specs.reputation + 1;

    return electedClass;
}

// Get the Kth nearest neighbor
function getKNN(subject, trainingData, K) {
    if (K === undefined) {
        K = knnConfig.K;
    }
    var distances = [];
    for (var i = 0; i < trainingData.length; i++) {
        var to = trainingData[i];
        for (var j = 0; j < to.instances.length; j++) {
            var traningInstance = to.instances[j];
            var distance = dtwClassifier.getEuclideanDistance(
                    subject.data,
                    traningInstance.data
                    );

            distances.push({
                'alias': to.item.alias,
                'specs': traningInstance.specs,
                'distance': distance
            });
```

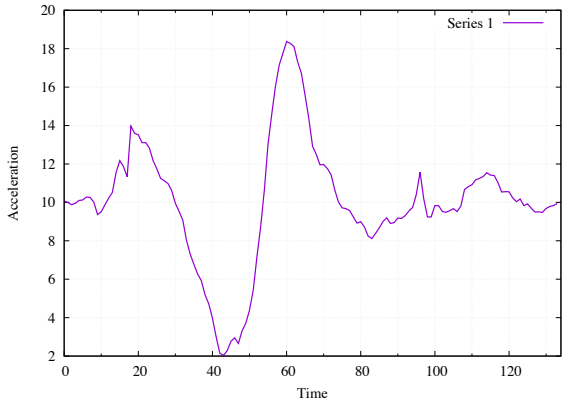
```
        }
    }

    // Sort the training instances based on their distance with the subject
    distances.sort(function(a, b){
        return a.distance - b.distance;
    });

    // Get the K nearest neighbour
    var candidates = [];
    for (var i = 0; i < K; i++) {
        candidates.push(distances[i]);
    }

    numberOfInquiries++;

    return candidates;
}

// Perform the cross validation of K-NN classification
function crossValidation(trainingData, voter, K, debug) {
    var characterBasedClassificationResult = {};
    var totalClassificationResult = {
        'correct': 0,  // Total number of correct classifications
        'incorrect': 0 // Total number of incorrect classifications
    };
    if (K === undefined) {
        K = knnConfig.K;
    }
    if (debug === undefined) {
        debug = true;
    }

    //console.log('f^: is a predicted class and f: is an ctual class');
    for (var i = 0; i < trainingData.length; i++) {
        var alias = trainingData[i].item.alias;
        characterBasedClassificationResult[alias] = {
            'correct': 0,
            'incorrect': 0
        };
        for (var j = 0; j < trainingData[i].instances.length; j++) {
            // Bring out the training instance (subject) from training set
            var subject = trainingData[i].instances.shift();

            // Use K-NN to estimate the classification of this subject
            var candidates = getKNN(subject, trainingData, K);
            var predictedClass = voter(candidates);
            var actualClass = trainingData[i].item.alias;
            //console.log('f^:' + predictedClass + ' f:' + actualClass);
            if (predictedClass === actualClass) {
                totalClassificationResult.correct++;
                characterBasedClassificationResult[alias].correct++;
            } else {
                totalClassificationResult.incorrect++;
                characterBasedClassificationResult[alias].incorrect++;
            }
            // Put training instance used as a subject back to our training set
            trainingData[i].instances.push(subject);
        }
    }
```

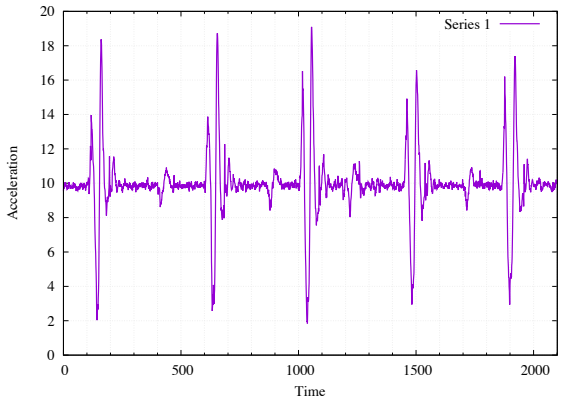
```
for (var i = 0; i < trainingData.length; i++) {
            var alias = trainingData[i].item.alias;
            // Print out classification accuracy result of each class 'A, B, ... Z'
            console.log(alias + ': ' + calculateAccuracy(
characterBasedClassificationResult[alias]) + ',');
        }
    }

    // Return total classification accuracy result
    return calculateAccuracy(totalClassificationResult);
}

// Calculate Accuracy based on the obtained result
function calculateAccuracy(result)
{
    return math.round((result.correct * 100)/(result.correct + result.incorrect));
}

exports.getKNN = getKNN;
exports.crossValidation = crossValidation;
exports.simpleVote = simpleVote;
exports.distanceBasedVote = distanceBasedVote;
exports.userBiasBasedVote = userBiasBasedVote;
exports.reputationBasedVote = reputationBasedVote;
```

Appendix 2 (Training instances)

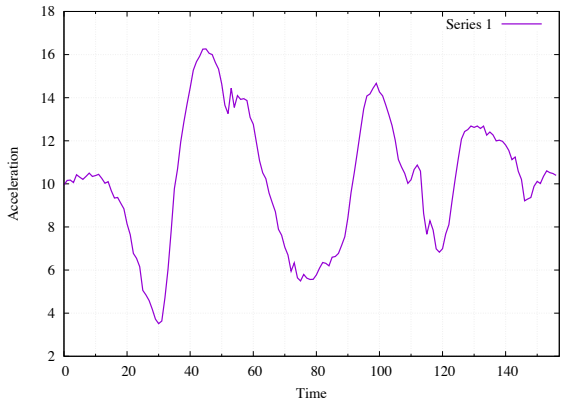
Figures in left shows only one training instance while figure in right shows 4 or 5 instances.

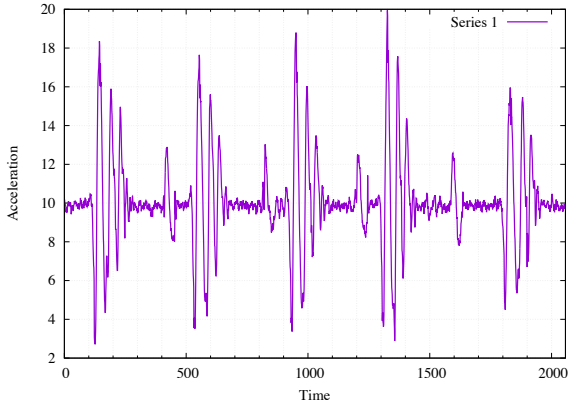
A



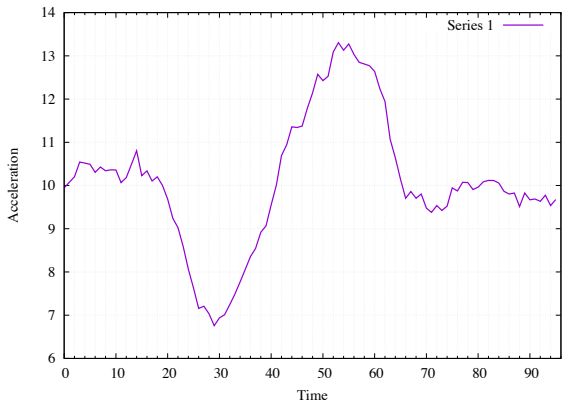


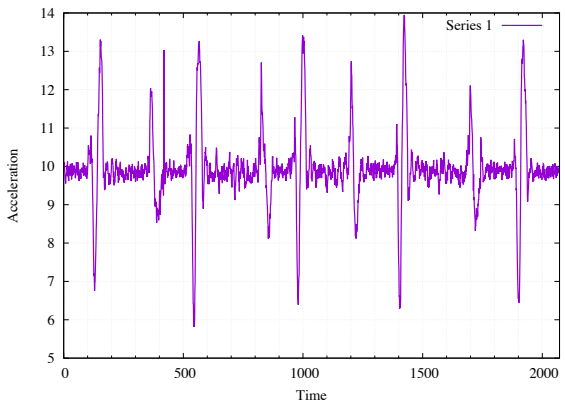
B



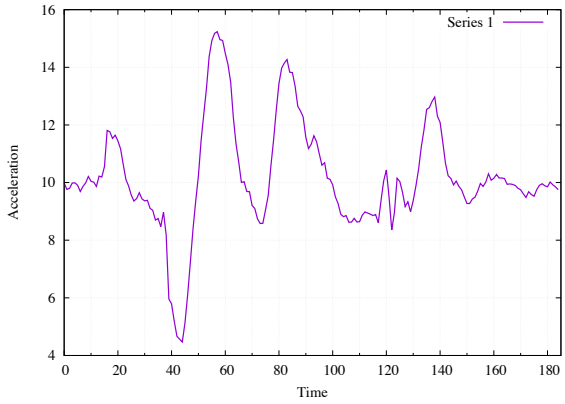


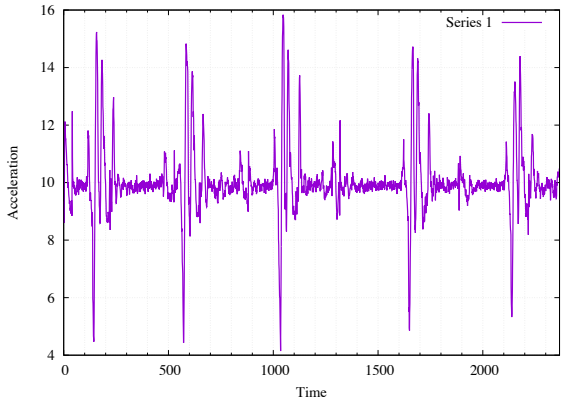
C



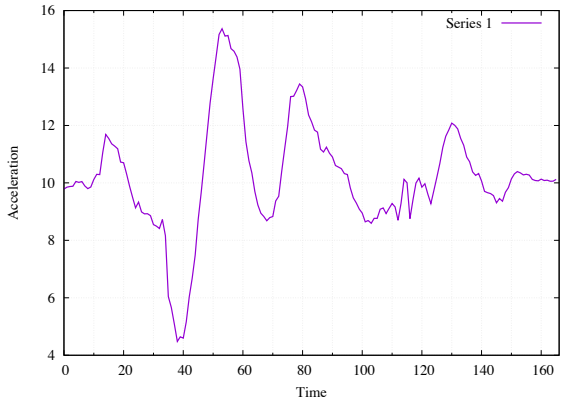


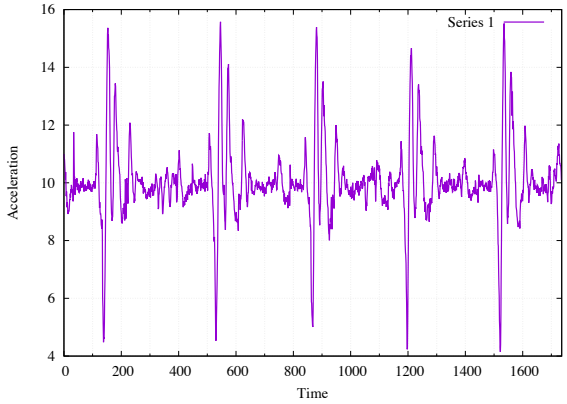
D


Series 1
Acceleration
Time
16
14
12
10
8
6
4
0
20
40
60
80
100
120
140
160
180

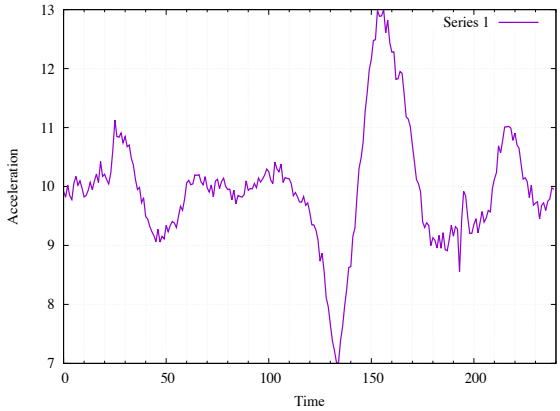

Series 1
Acceleration
Time
16
14
12
10
8
6
4
0
500
1000
1500
2000

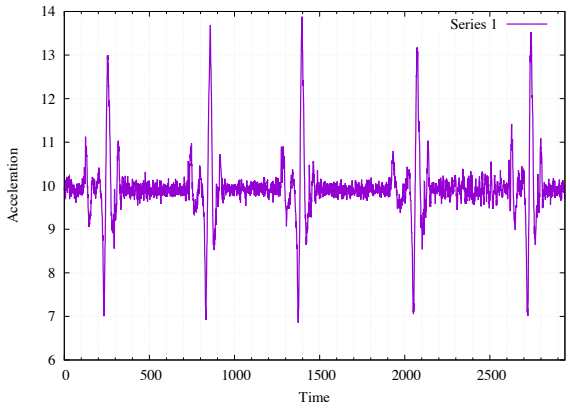
E


Series 1
Acceleration
Time
16
14
12
10
8
6
4
0
20
40
60
80
100
120
140
160

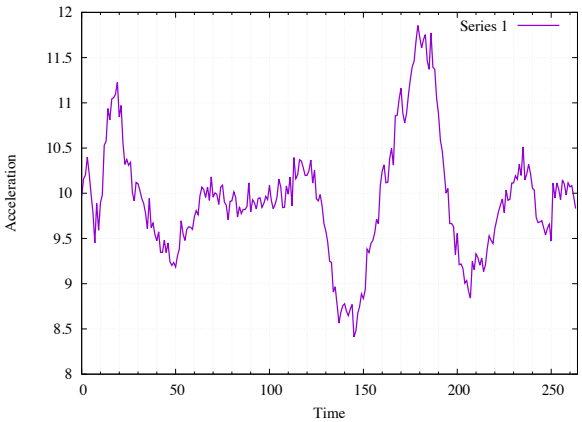

Series 1
Acceleration
Time
16
14
12
10
8
6
4
0
200
400
600
800
1000
1200
1400
1600

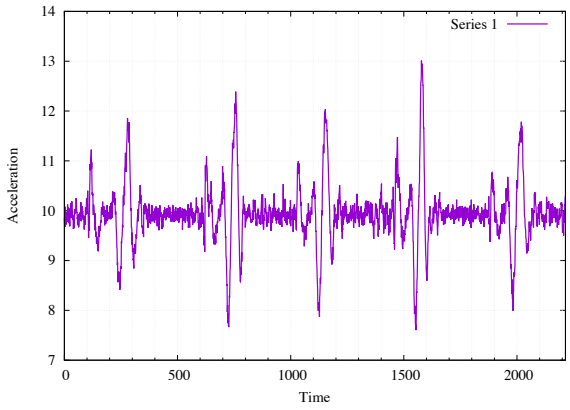
F


Series 1
Acceleration
Time
13
12
11
10
9
8
7
0
50
100
150
200

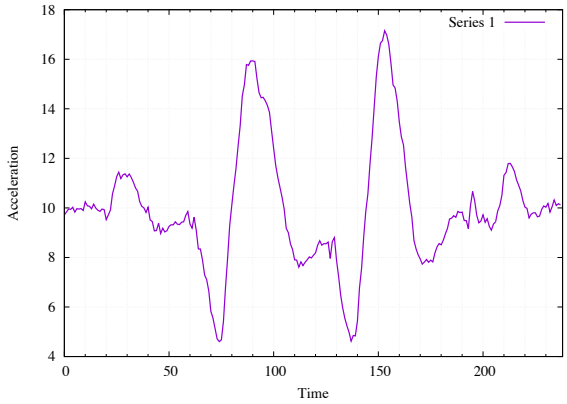

Series 1
Acceleration
Time
14
13
12
11
10
9
8
7
6
0
500
1000
1500
2000
2500

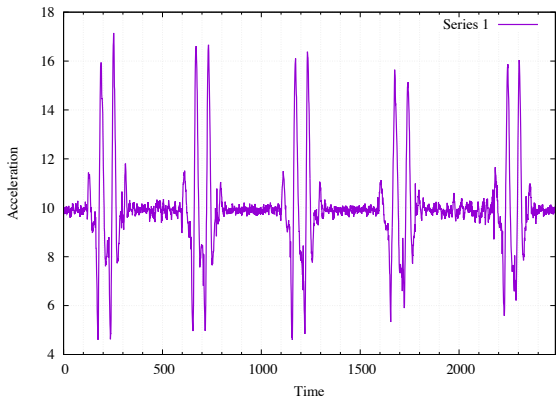
G


Series 1
Acceleration
Time
12
11.5
11
10.5
10
9.5
9
8.5
8
0
50
100
150
200
250

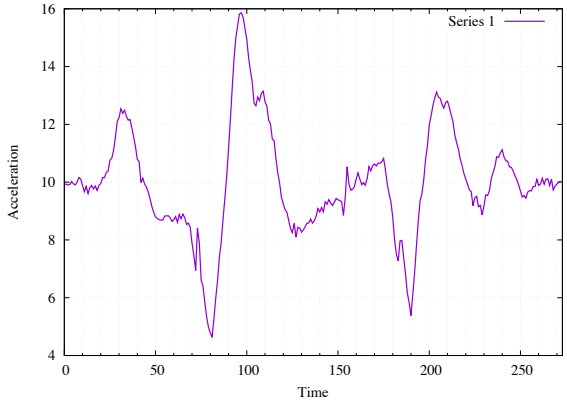

Series 1
Acceleration
Time
14
13
12
11
10
9
8
7
0
500
1000
1500
2000

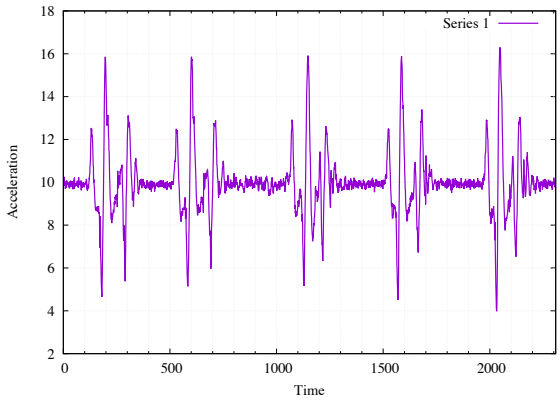
H


Series 1
Acceleration
Time
18
16
14
12
10
8
6
4
0
50
100
150
200

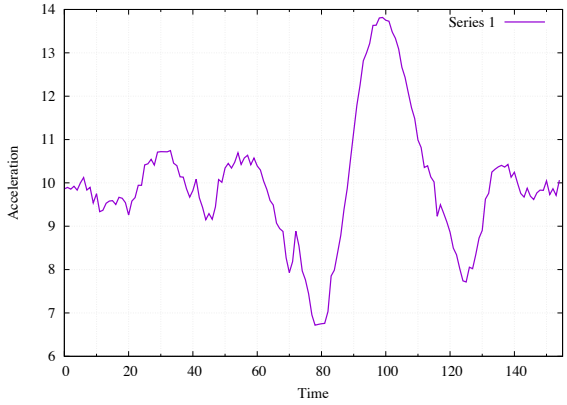

Series 1
Acceleration
Time
18
16
14
12
10
8
6
4
0
500
1000
1500
2000

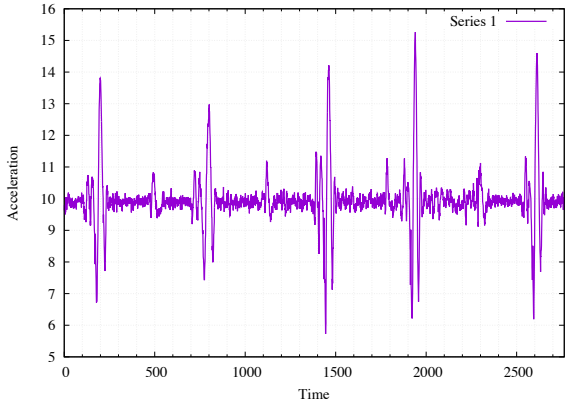
I


Series 1
Acceleration
Time
16
14
12
10
8
6
4
0
50
100
150
200
250

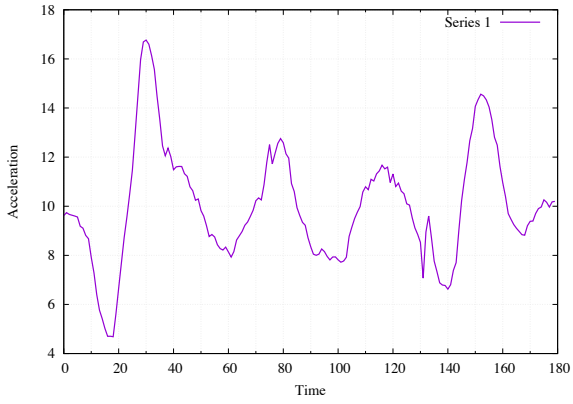

Series 1
Acceleration
Time
18
16
14
12
10
8
6
4
2
0
500
1000
1500
2000

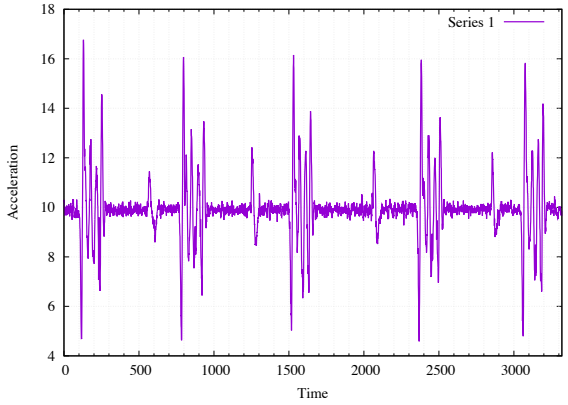
J


Series 1
Acceleration
Time
14
13
12
11
10
9
8
7
6
0
20
40
60
80
100
120
140

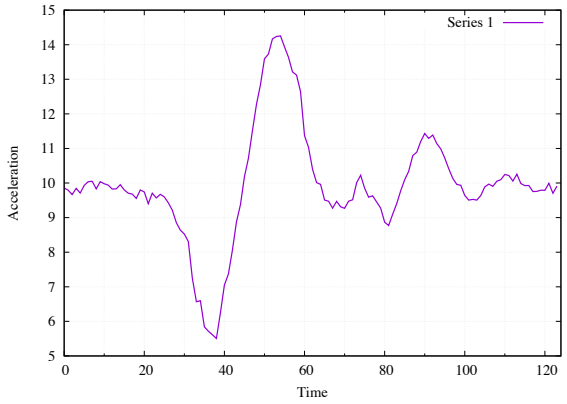

Series 1
Acceleration
Time
16
15
14
13
12
11
10
9
8
7
6
5
0
500
1000
1500
2000
2500

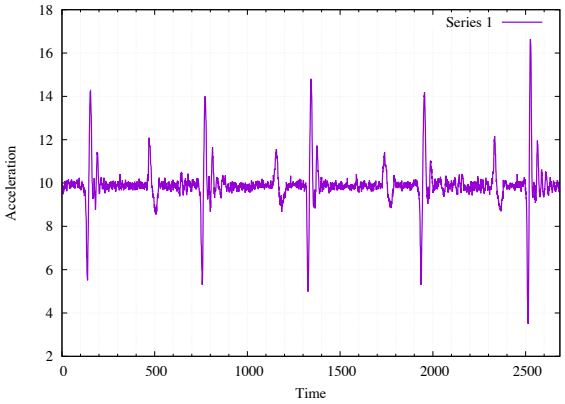
K


Series 1
Acceleration
Time
18
16
14
12
10
8
6
4
0
20
40
60
80
100
120
140
160
180

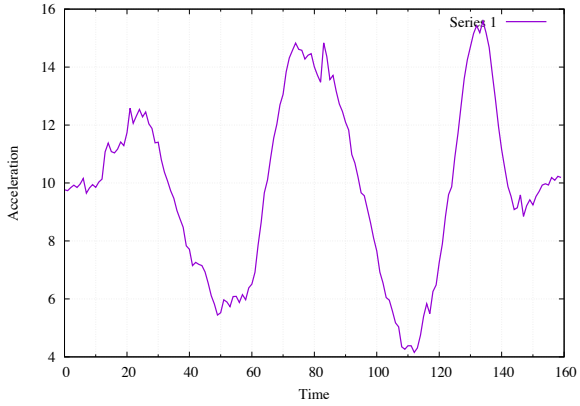

Series 1
Acceleration
Time
18
16
14
12
10
8
6
4
0
500
1000
1500
2000
2500
3000

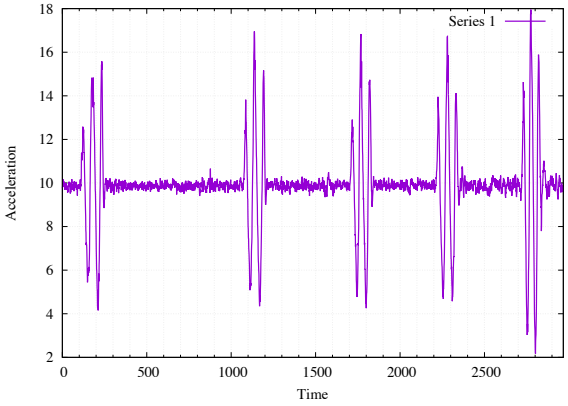
L


Series 1
Acceleration
Time

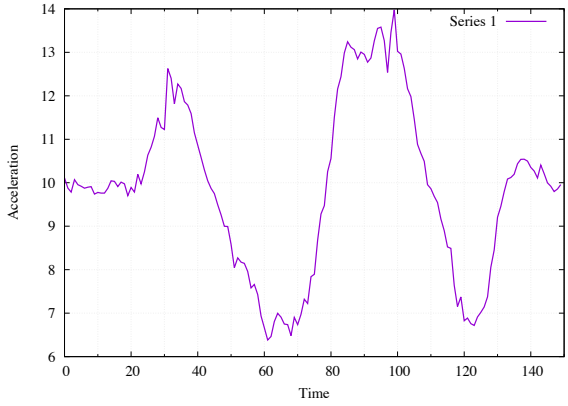

Series 1
Acceleration
Time

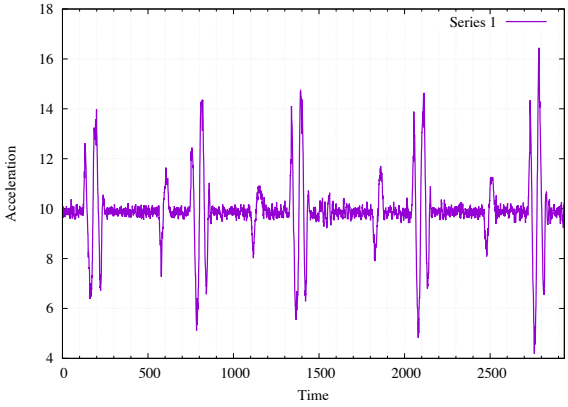
M


Series 1
Acceleration
Time

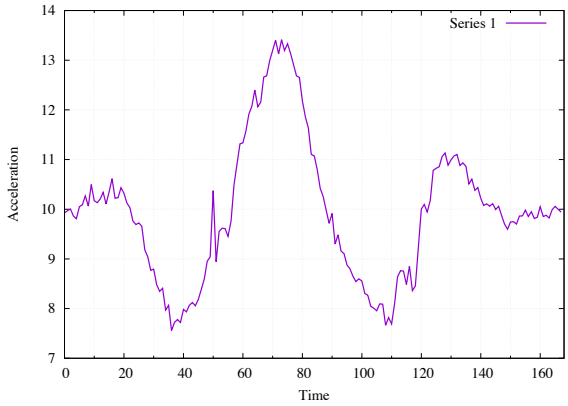

Series 1
Acceleration
Time

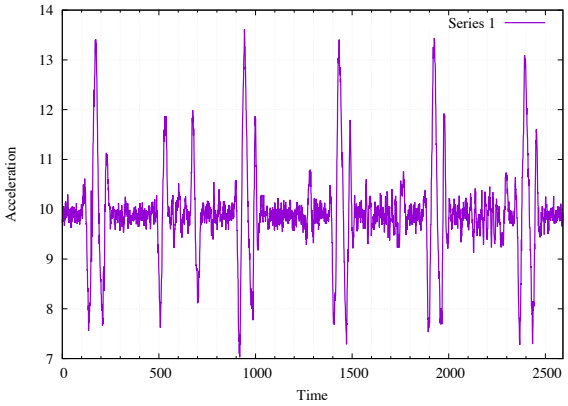
N


Series 1
Acceleration
Time

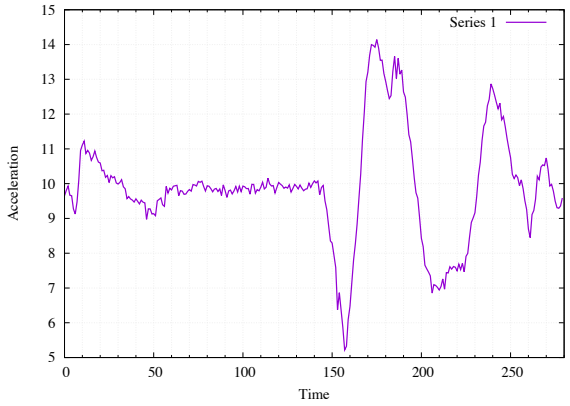

Series 1
Acceleration
Time

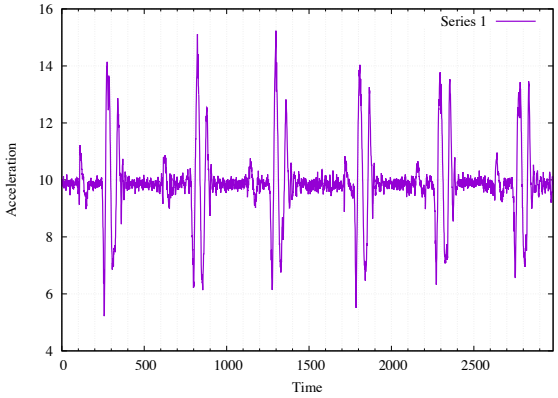
O


Series 1
Acceleration
Time

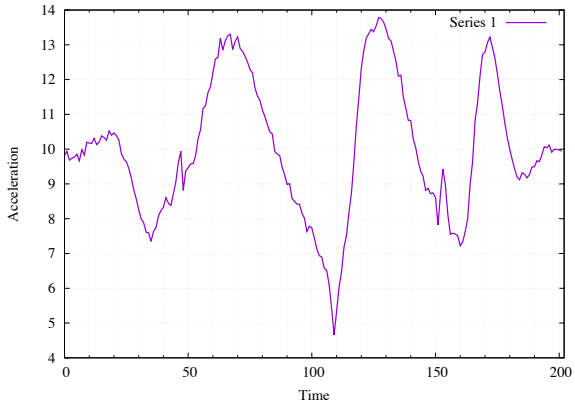

Series 1
Acceleration
Time

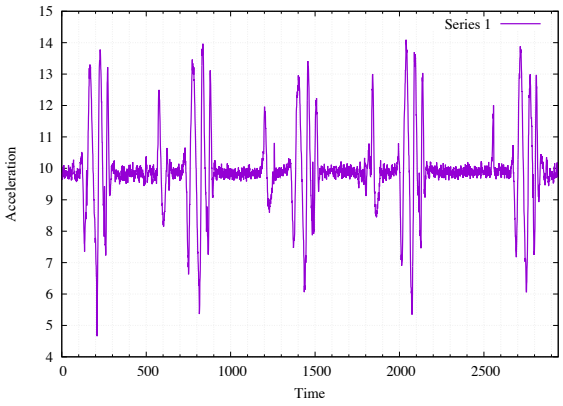
P


Series 1
Acceleration
Time

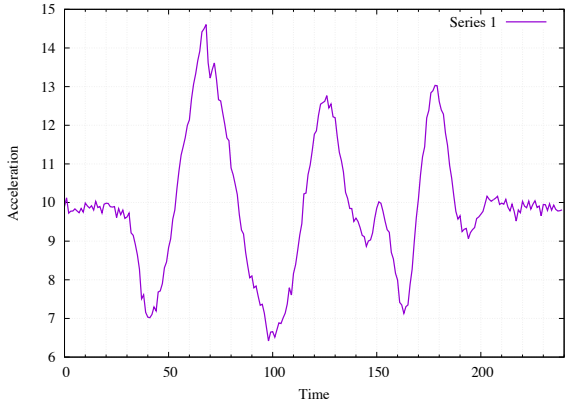

Series 1
Acceleration
Time

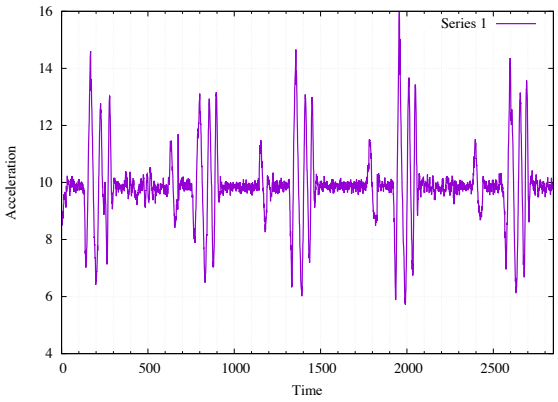
Q


Series 1
Acceleration
Time

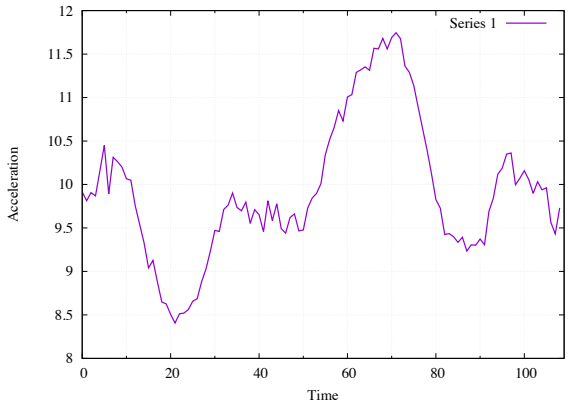

Series 1
Acceleration
Time

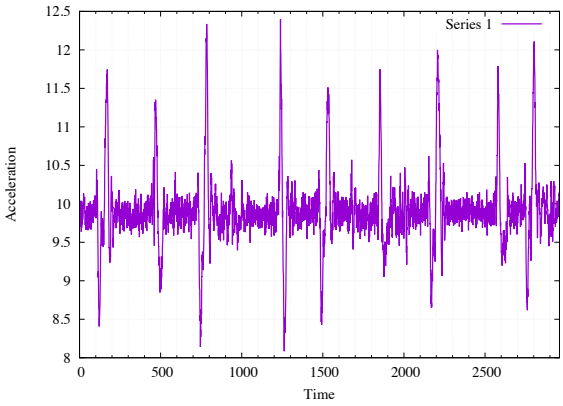
R


Series 1
Acceleration
Time

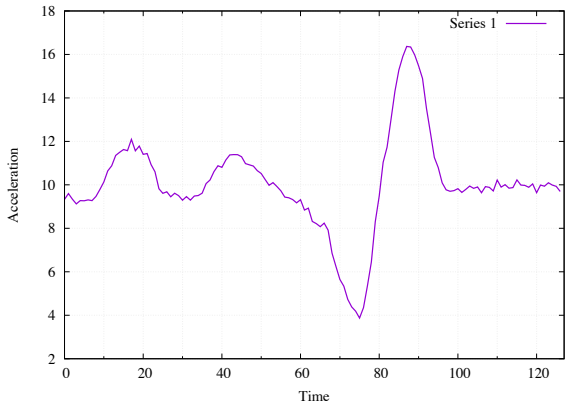

Series 1
Acceleration
Time

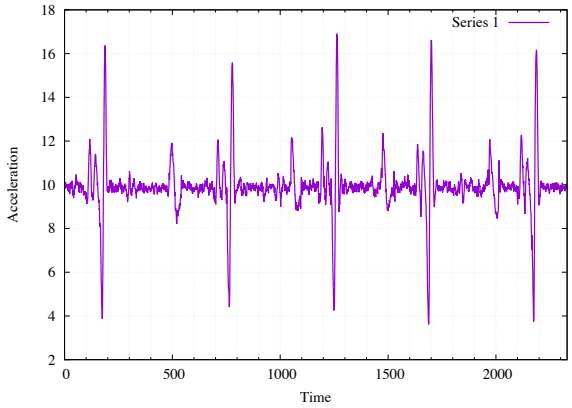
S


Series 1
Acceleration
Time

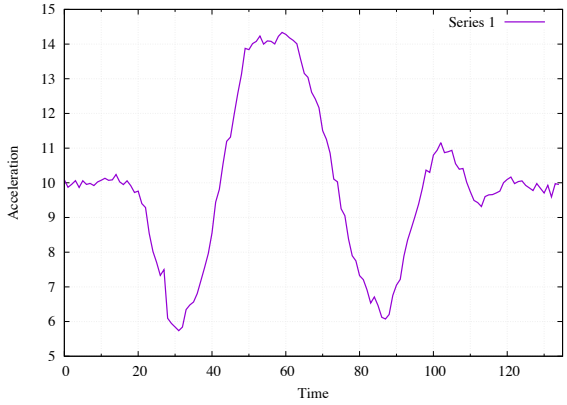

Series 1
Acceleration
Time

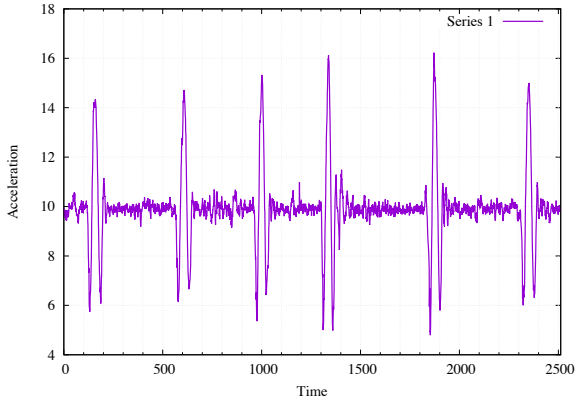
T


Series 1
Acceleration
Time

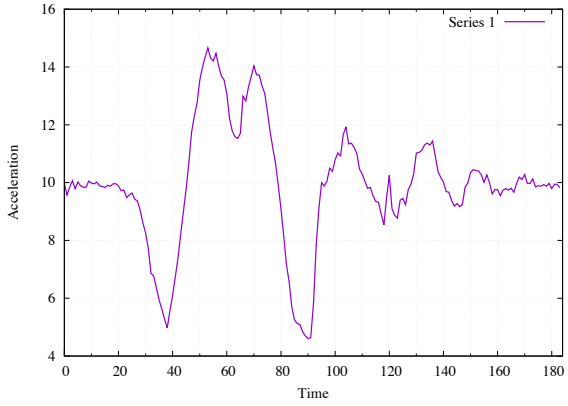

Series 1
Acceleration
Time

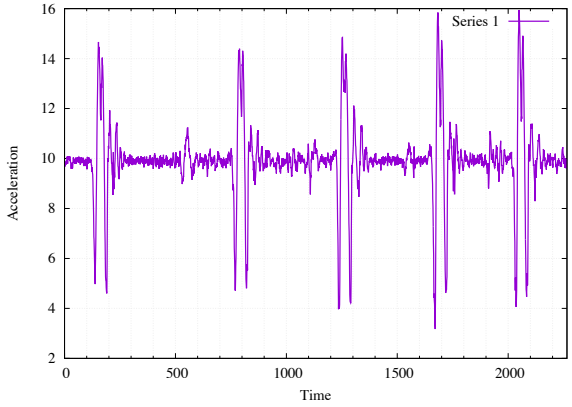
U


Series 1
Acceleration
Time

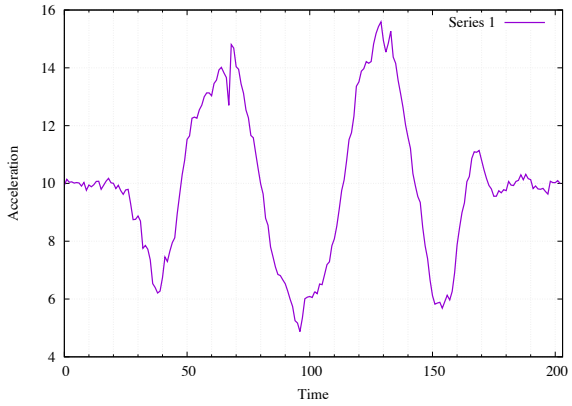

Series 1
Acceleration
Time

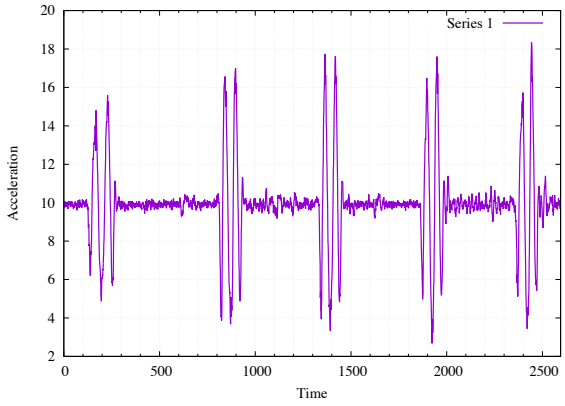
V


Series 1
Acceleration
Time

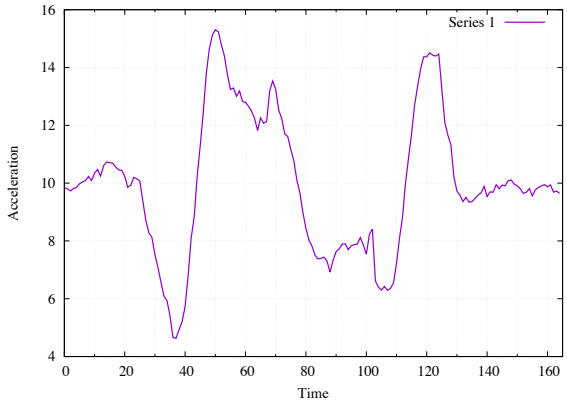

Series 1
Acceleration
Time

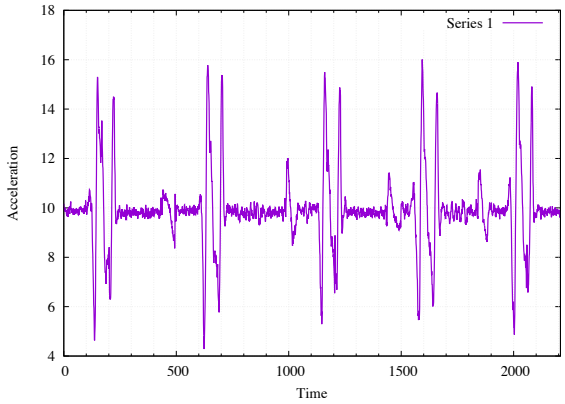
W


Series 1
Acceleration
Time

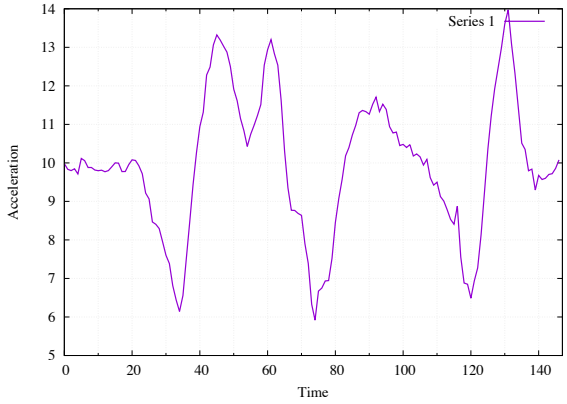

Series 1
Acceleration
Time

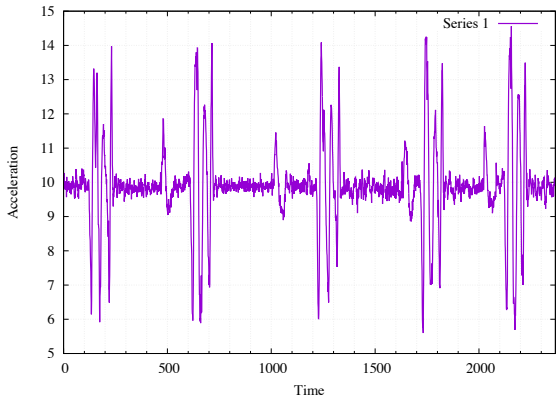
X


Series 1
Acceleration
Time
16
14
12
10
8
6
4
0
20
40
60
80
100
120
140
160

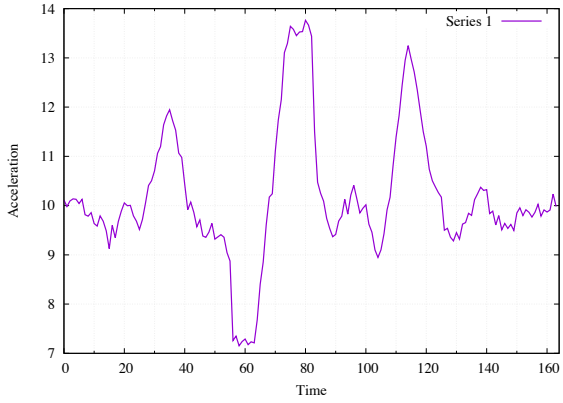

Series 1
Acceleration
Time
18
16
14
12
10
8
6
4
0
500
1000
1500
2000

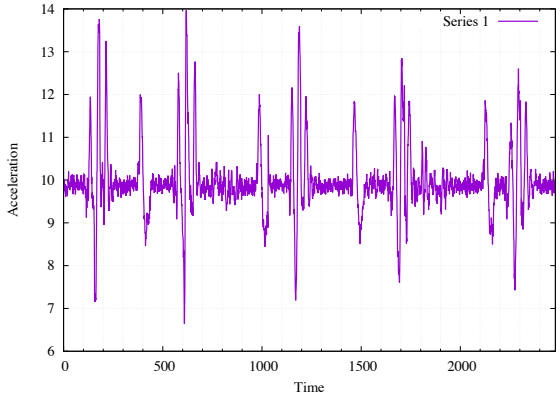
Y


Series 1
Acceleration
Time
14
13
12
11
10
9
8
7
6
5
0
20
40
60
80
100
120
140


Series 1
Acceleration
Time
15
14
13
12
11
10
9
8
7
6
5
0
500
1000
1500
2000

Z


Series 1
Acceleration
Time
14
13
12
11
10
9
8
7
0
20
40
60
80
100
120
140
160


Series 1
Acceleration
Time
14
13
12
11
10
9
8
7
6
0
500
1000
1500
2000

References

[1] J. Chang. (2010). “A New Way to Interact with the Cloud,” *Microsoft Research Lab*. [Online]. Available: http://research.microsoft.com/en-us/news/features/030419-clientcloud.aspx.

[2] Eugster, Patrick, Vinaitheerthan Sundaram, and Xiangyu Zhang. "Debugging the Internet of Things: The Case of Wireless Sensor Networks." Software, IEEE 32.1 (2015): 38-49.

[3] Shimmer, “Shimmer Research Group.” [Online]. Available: http://www.shimmersensing.com/.

[4] ZigBee, “ZigBee Alliance.” [Online]. Available: http://www.zigbee.org/.

[5] Ferraris, F., Grimaldi, U., Parvis, M.: Procedure for Effortless In-Field Calibration of Three-Axis Rate Gyros and Accelerometers. Sensor and Materials 7, 311–330 (1995).

[6] Shimmer 9DOF Calibration User Manual Rev 1.0b, 2011 Shimmer Research.

[7] Reiss, Attila, et al. "Activity recognition using biomechanical model based pose estimation." Smart Sensing and Context. Springer Berlin Heidelberg, 2010. 42-55.

[8] Ermes, Miikka, Juha Pärkkä, and Luc Cluitmans. "Advancing from offline to online activity recognition with wearable sensors." Engineering in Medicine and Biology Society, 2008. EMBS 2008. 30th Annual International Conference of the IEEE. IEEE, 2008.

[9] Pohl, Henning, and Aristotelis Hadjakos. "Dance pattern recognition using dynamic time warping." *Sound and Music Computing* 2010 (2010).

[10] https://github.com/langholz/dtw

[11] https://github.com/SamanShafigh/vWrite