Detecting Harmful Hand Behavior with Machine Learning from Wearable Motion Sensor Data

by

Lingfeng Zhang

A thesis submitted to the Graduate School of
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
December, 2015

We the undersigned committee hereby approve the attached thesis, "Detecting Harmful Hand Behavior with Machine Learning from Wearable Motion Sensor Data" by Lingfeng Zhang.

_____

Philip Chan, Ph.D.
Associate Professor and Major Advisor
Computer Sciences and Cybersecurity

_____

Debasis Mitra, Ph.D.
Professor
Computer Sciences and Cybersecurity

_____

Joshua Pritchard, Ph.D.
Assistant Professor
School of Behavior Analysis

_____

Richard Newman, Ph.D.
Professor and Department Head
Computer Sciences and Cybersecurity

Abstract

Title: Detecting Harmful Hand Behavior with Machine Learning from Wearable Motion Sensor Data

Author: Lingfeng Zhang

Advisor: Philip K. Chan, Ph.D.

In medical care and special needs areas, human activity recognition helps doctors track the patients while they are unsupervised. In this paper, we will present our classifier system for detecting harmful hand behavior. The data comes from a wearable sensor on the user's wrist. It collects signals in the three axes x, y and z. For each axis, it contains multiple attributes. Because reducing irrelevant attributes can decrease the time complexity and increase the accuracy, we started processing the raw data by ignoring some attributes from the whole attributes set. Our design approach is to apply a classification algorithm which generate an initial output, and then use a sequence post process to correct potentially incorrect initial outputs. The basic classifier algorithm is a decision tree, where we adopt the random forest approach to reduce generation error. Experimental results show that the system can get a 96% accuracy rate in detecting harmful behavior, and it can also obtain 95% accuracy rate distinguishing the ambiguous behaviors from the harmful behaviors.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I want to thank my advisor Dr. Philip Chan, he is so patient and responsible to me. During this one year research with him, I learnt a lot of knowledge in machine learning field. It will be the precious memory in my life. I felt lucky to be one of his students.

To my Committee Members, Dr. Mitra and Dr. Pritchard, thank you for taking the time to read and revise my thesis.

I also want to thank my friends and classmates, Markoto Morri, Huizhong Hu, Kai Wang, Xunhu Sun, they accompany with me in my master degree and we had a wonderful time during the past two yeas. Thank you all of you!

Chapter 1
Introduction

Human Activity Recognition is becoming more and more popular because it attracts a lot of researchers and it was a strong potential for assistant services, such as sports tracking, health care, special needs, and other routine actions. In order to build a system to recognize human activities, people need to get information from individuals, which means recording their human activity history and using this history data is essential. Video recorders, cameras, microphones, and accelerometers are the most frequently used devices to obtain data. Wearable accelerometers are widely deployed in many cases, because they are convenient and suitable for the user, and they are available for long term tracking.

Several studies have showed significant improvement in activity recognition, but most of these focused on the body movement scenarios, such as walking, running, sitting, etc. In this paper, we are going to present a comparable method to recognize self-injuring behavior with autistic children.

There were some previous studies[9,10] working on the hand behavior recognition, but our target is detecting some abnormal behaviors such as aggressive attacks toward the head or knees. As we know, the target anomalous data occurs much less than the normal data, therefore for machine learning purpose we needed to first solve the imbalance data set before building the system.

In this paper, we designed the decision tree algorithm as the main classifier and using a post sequence process to improve the accuracy. The raw data was acquired from a single wearable accelerometer located on the user's wrist. We did not use the raw data to do classification. Before that, we needed to do a windowing process, using each window as

input instance. After we got the windows, we extracted the features from each window[1]. An enormous set of features can lead to computational problems and may potentially decrease the accuracy. Therefore, a small and efficient set of features played an important role in our case. A Random Forest was implemented, because using Random Forest would result in a higher accuracy with a voting strategy. We also designed a Sequence Post-Processing method, where we used the outputs from the Random Forest as the inputs. The basic idea was to find the pattern information from the training set, then use this information to correct the wrong classifications from the Random Forest.

According to the results, we got an over 96% true positive rate of slapping class. This gave us the confidence in detecting the harmful hand behaviors. Besides this achievement, we proved that using the best combination of forest size and subset-feature size for Random Forest could obtain a outstanding performance; and the combinations for different data sets are not the same. Furthermore, we decreased the false positive rate by using Sequence Post-Processing. Even though there are some limitations in Sequence Post-Processing, as far as now it is still showed a good effect on out system.

The remaining sections of this paper are organized as follows: in Chapter 2, we will review some previous studies, and discuss some good ideas. In Chapter 3, we will present the data processing, system schema, and all the details about how to make the system work. The experimental results and data analysis is presented in Chapter 4, and the conclusion is in the last part Chapter 5.

Chapter 2
Related Work

2.1 Previous Work

There are 2 types of Human Activity Recognition studies: the first is body movement recognition, such as running, walking, sitting, and other basic motions, and the second is hand behavior recognition, where activities such as hand-waving, drinking, and slapping are more closely analyzed. Paper[2] shows the recent research on human activity recognition which is based on the accelerometer, and lists the techniques used in this research.

Some of the studies[1,3,4,5,6] focus on the body movement recognition. Papers[1,3] implemented the Hidden Markov Model(HMM) as their classification algorithm, and they did experiments to prove that a good sampling rate can improve the accuracy. For paper[4], the authors realized Singular Value Decomposition (SVD) which is a semantic indexing method attempting to capture the underling relationship among the features. SVD is helpful for discriminating between the classes and a good dimension reduction technique. They implemented the Back Propagation Neural Network(BPNN) as their classification algorithm. As for paper[5], the authors used the Principal Component Analysis (PCA) to reduce the complexity for features, and designed a divide-and-conquer method to separate the dynamic and static states. Finally, they used the BPNN to do the classification. The paper[6] used the Naive Bayes as the classification algorithm, which assumes that the features have strong independence with each other in one single instance.

The papers[8,9,10] focus on the hand behavior recognition. In paper[8], the authors computed the features from the input video images which are 2D positions of an user,

while implementing the Finite State Machine(FSM) to be the classification algorithm. Paper[9] implemented the HMM. The recognition system has a pre-processing step which removes the effect of device orientation from the data. In paper[10], the authors implemented Support Vector Machine(SVM) which is a small sample size method based on statistic learning theory, and compared the performance in two ways: user-dependent and user-independent.

The data for human activity recognition can also come from multiple sensors resources, or other resources instead of accelerometer. Paper[7] acquired the data from accelerometer sensor and single grid-based image sensor. The data in different channels will be processed by SVM, and the result shows a great improvement by using fusion sensors. In paper[11], there are two sensors: one of them records the accelerometer, and the other one records sound signal. Then, the authors used the Linear discriminant analysis to deal with the sound channel and the HMM to deal with the acceleration data for the activities classification. Paper[12] used a real time continuous video stream to be the data, while BPNN was implemented to do the classification.

2.2 The difference between this paper and previous work

This paper will try to distinguish between different types of hand behaviors. Even though some previous research studied hand behavior recognition, their research is about normal behaviors, such as waving the hand, making a hand sign, etc. But for this paper, the target is to classify all the harmful hand behaviors from some normal behaviors.

Basically, the data for this paper came from a wearable sensor which contained accelerometer and gyroscope in three dimensions x, y, and z. We focused on the hand behavior, ignoring the body movement. We simulated the autistic children's behaviors, which means the bad hand behaviors only happened a few times during a certain period time. Moreover, we introduced another behavior - drinking water, which is to test whether our system can distinguish these two similar behaviors.

As we know, harmful hand behaviors lasted a shorter time than the time when there were no harmful hand behavior. Because the wearable sensor will collect the data within a certain amount of time, therefore the data of harmful behaviors will be much less than the data of non-harmful behaviors. This is called an imbalance data set. As to imbalance data set, accuracy for the whole test set is not helpful for us to evaluate our system. Instead of that, the true positive rate and the false positive rate will be the criteria.

Generally speaking, in this paper, our job is to recognize both harmful and non-harmful behaviors, while the data comes from the normal user who stimulates the autistic child's behaviors. There is no orderliness for the training data. Based on these situations, we will design a system to do the classification which can get both a high true positive rate and a low false positive rate.

Chapter 3
Approach

The main problem in this paper is to detect harmful actions from motion sensor data. In this chapter, we proposed the main components of the overall system. First of all, data pre-processing was necessary before starting the machine learning process. After processing the raw data, there were two algorithms which would be implemented to complete the classification. The first algorithm was Random Forest, which used a voting strategy to avoid noise and obtain a good performance. The second algorithm was called Sequence Post-Processing. The goal of Sequence Post-Processing was to correct some incorrect classifications, which come from the first algorithm. Here is a brief schema in Figure 3.1.

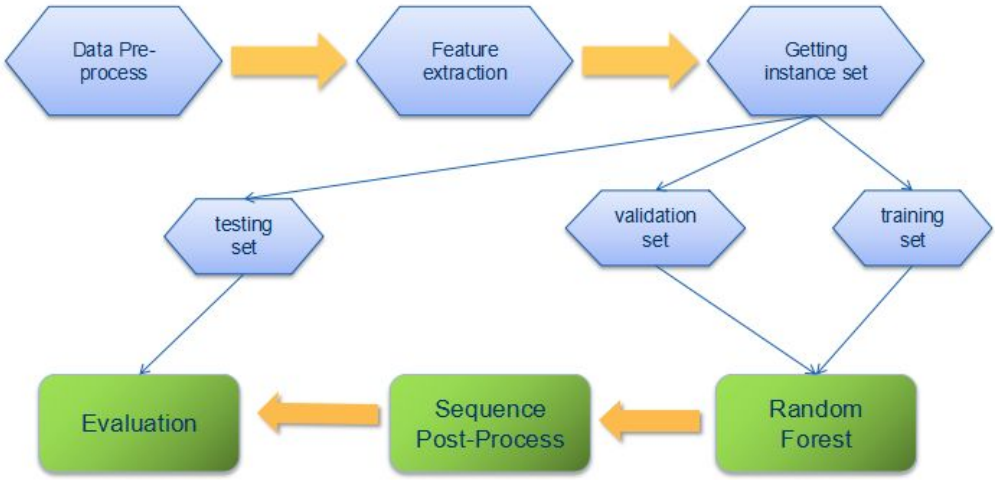


Figure 3.1 — The basic components of the overall systems.

3.1 Data Pre-processing and Feature Extraction

The raw data came from the wearable sensor, which included the accelerometer and the gyroscope in x, y, and z dimensions. The unit for the accelerometer is gravity. We used the Linear Acceleration, which excludes Earth's gravitation, but only counts the object's acceleration. The unit for the gyroscope is degree per second. The data was recorded by the sensor every 10 milliseconds, which means that in 1 second there are around 100 records. Figure 3.2 shows a part of the raw data.

SensorId	TimeStam	FrameNum	GyroX (deg/s)	GyroY (deg/s)	GyroZ (deg/s)	LinAccX (g)	LinAccY (g)	LinAccZ (g)
0	0	0	0.38274	-1.443478	0.538024	-0.003341	0.003274	0.04996
0	0.009995	1	-0.107167	-1.513465	0.887958	-0.005175	0.00321	0.040221
0	0.019989	2	0.102793	-2.423293	0.118103	0.000868	0.016728	0.046126
0	0.029999	3	-0.387115	-2.143346	0.048116	-0.004691	-0.002809	0.055941
0	0.039993	4	-0.667062	-1.933386	0.468037	-0.004472	0.001116	0.050115
0	0.049988	5	-0.387115	-1.863399	0.328063	-0.002493	0.002932	0.044275
0	0.059998	6	0.312754	-1.723425	-0.231831	-0.000368	-0.008382	0.057931
0	0.069992	7	0.452727	-1.443478	0.048116	-0.002215	-0.004196	0.042294
0	0.080002	8	0.102793	-2.633254	0.18809	0.005802	-0.005948	0.050117
0	0.089996	9	-0.387115	-1.583452	0.677997	0.007917	-0.015494	0.046213

Figure 3.2 — The example of raw data

Dr.Pritchard in School of Behavior Analysis at Florida Institute of Technology provided these data. One of his goals is to design therapies to help autistic children, who sometimes perform harmful actions to themselves. To evaluate the effectiveness of his therapies, he would like to count the number of actions in an automated manner, instead of having someone count manually. This project tries to detect harmful hand behaviors from motion sensor data.

3.1.1 Data pre-process

The given data set was a period of real time record. The data included some harmful behavior actions, such as slapping head, and several normal behavior actions, including drinking water. Besides this given data set, we had another action table which contains the time label to indicate the starting/ending time for each action.

The time label might be less accurate, because it used the integer second to be the unit. But as we mentioned before, the sensor collects the data in each 10ms, so usually the actual actions' starting/ending time may be earlier or later than the given time label. Therefore, according to the approximate time, we plot the data in a visual way. Then, we manually set a more accurate starting/ending time label, which kept the time in two decimal places.

After this process, each data record in the raw data set belonged to different classes, such as slapping and drinking; these are behavior actions that are either harmful or normal. For the rest of data, we regarded them as "no action" class. We called the behavior classes and the "no action" class as positive classes and negative class respectively. Since we had the data in different classes, the criteria to evaluate our system shows whether our system can recognize the data in the correct class.

3.1.2 Windowing the Data

One single data does not contain enough information for our machine learning system. Therefore, we need to window the data, so that we can extract some useful information/features from a group of data. If the window size is too small, which means the data is too few to represent an action, the feature can not be completely extracted. On the contrary, if there is too much data in one window, some irrelevant data may be included. The irrelevant data can be the noise in the window, and the accuracy of the system can be influenced in terms of this reason.

Considering these situations, we evaluated the entire data set and found the action which has the smallest number of data records. Then set the window size as this number. Firstly, this window size is enough to express an action. Secondly, this window size is not too large where it may contain other irrelevant data.

In our data set, the window size is 25. After we window the whole raw data set, we need to sign a class to each window. If over 50% of the data records have the same class X in one

window, then we sign class X to this window. We keep on doing this process for all windows.

In our case, there are 25 records in one window, which means there are around a quarter of second records in one window. We think that if we want to predict what the current behavior action is, maybe the information of the previous actions can help us. So we grouped four continuous windows to be one instance, and use the last window's class to be the class of this instance. Four windows can represent 1 second information. Figure 3.3 indicates how an instance is formed.

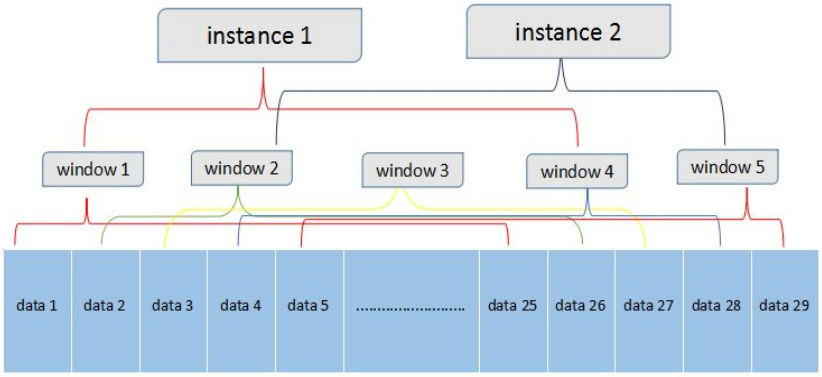


Figure 3.3 — Each window has 25 raw data, each instance has 4 windows

3.1.3 Feature Extraction

According to the previous researches[2,5], we got some ideas about choosing the features. In this paper, we will introduce seven features, which are: (1) mean value; (2) the absolute value between min and max: the difference between min and max is useful in discriminating whether there is an action happening now or not; (3) root mean square value: this feature is to describe the distribution of the data in one period; (4) standard deviation; (5) linear regression: linear regression is to represent the trend of a group data; (6) pearson correlation between axes: pearson correlation can represent the relation in two dimensions,

either positive related or negative related; (7) pearson correlation between accelerometer and gyroscope in the same axis.

The first six features are extracted from the accelerometer and gyroscope in three axes; therefore, there are 6*2*3=36 features. The 7th feature-pearson correlation between the accelerometer and gyroscope is only available in the same axis, so there are 3 features more. Finally, there are 39 features in total in one window.

3.1.4 Separate the data for training and testing

After the feature extraction, each instance will contain 39*4=156 features(each window contains 39 features and 1 instance has 4 windows). Now, we can use the instances to be the input for the machine learning training process. In order to evaluate our system later on, we need to set a test set from the whole instance set. We randomly choose a continuous block, which is one-fourth of the whole instance set, to be the test set. The instance in the test set will not be trained, but it will only be used for the evaluation process. So, the remaining instances will be the training set. We will also randomly choose one-third of the instances from the training set to be the validation set; that is for the post-sequence processing.

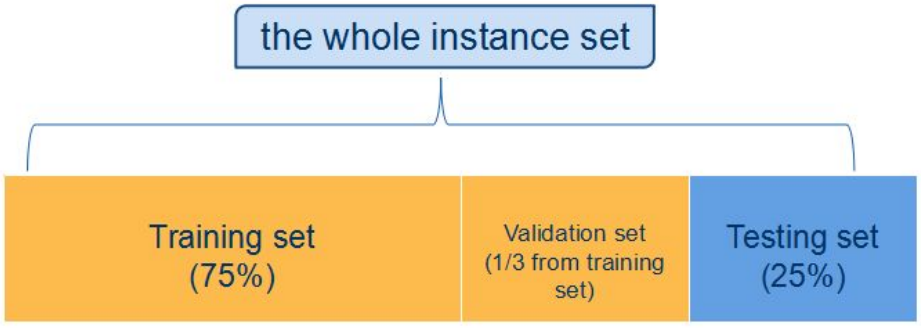


Figure 3.4 — The testing set is 1/4 of the whole set and validation set is 1/3 of the training set.

3.2 Learning Algorithm and Imbalance Data Set

In this paper, we implemented the Decision Tree algorithm as the main learning algorithm. Our given data set was an imbalanced data set, because the number of negative class instances was much larger than the number of positive class instances. Therefore, before the machine learning process, we needed to deal with the imbalance data set. Based on the decision tree algorithm, we implemented the Random Forest algorithm. Random Forest produced a better accuracy and handle a high number of features. Another advantage of Random Forest was that it was less likely to be over-fitted in the training process.

3.2.1 Decision Tree Learning

Decision Tree is a tree structure classification algorithm, where each internal node represents a feature, each branch represents the value of the feature, and the leaf node represents a class label. So, the decision is being made at each internal node(or feature), the instances which have the same value of the feature will go to the same branch. The decision taken after computing all features will be dependent on whether or not the instances in the same leaf node are the same class or most of the instances have the same class. Figure 3.5 is an example tree to show how the decision tree works:

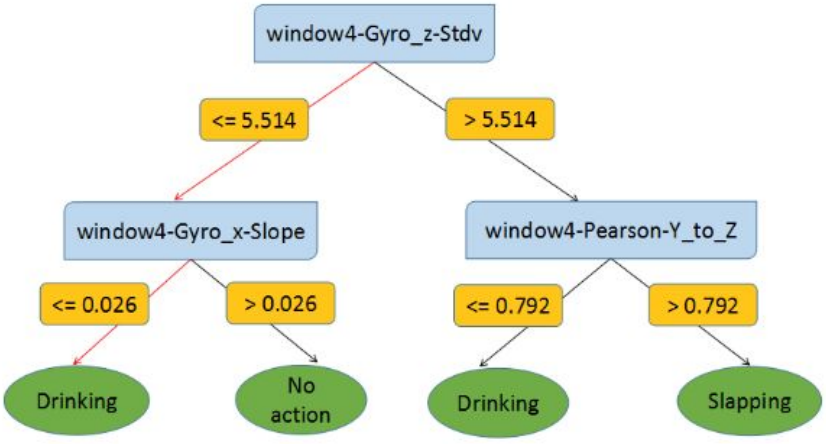


Figure 3.5 — The blue block is the feature. The yellow block is the value of each feature. The green circle is the leaf node represents the class

The key point of the decision tree is how to select a feature at each internal node, thus, making the next sample space less confusing. We use entropy to describe the sample space:

$$Entropy(P(v_1),...,P(v_n)) = \sum_{i=1}^{n} -P(v_i) * \log_2 P(v_i) \quad (3.1)$$

Where v_1 to v_n represent the different classes at a current sample space. P(vi) is the fraction of v_i class in the current sample space. A smaller entropy represents a more pure sample space. So, we will have an entropy in some current level, we say it is $Entropy(current)$, and we will have different entropy in the next level. After selecting the different feature, we say it is $Entropy(feature_i)$. Therefore, we can get an information gain $Gain(feature_i)$ which is:

$$Gain(feature_i) = Entropy(current) - Entropy(feature_i) \quad (3.2)$$

We picked the feature which has the highest gain and set it to be the internal node, continuously, until we finish building the entire tree.

3.2.2 Imbalance Data Set

In general classification problems, the data of different classes is approximately equally represented. But, for the imbalanced data set, the class distribution is not uniform among the classes. In our situation, the data of slapping class was extremely rare compared with data of no action class; the data of drinking class was also much less than the data of no action class. We call the minority class as positive class and the majority class as negative class.

There are three issues with the imbalanced data set. The first issue is that, since the negative class is larger than the positive class, according to the decision tree learning, maybe all the leaf nodes are negative class, because the negative class usually wins in a voting process when a leaf is not pure. The second issue is that, because the number of

positive classes is very small, it is easy to get some super features that can cover these positive classes. These super features are only effective in these particular positive class instances; however, in the actual application, these super features do not contain enough information to identify the positive class instance. The third issue is that the evaluation system can obtain a high accuracy by simply classifying all the instances as negative class. Missing all the positive classes but still getting a high accuracy, which is not what we expect.

There are several ways to solve the imbalance data problem. Here we discussed the method we used. Firstly, we selected all the instances which are positive classes (slapping and drinking) and remember the number of these instances, let's mark it as *K*. Secondly, we randomly selected *K* instances which were negative class. Finally, we mixed the *K* positive class instances and the *K* negative class instances together; therefore, we had a 2*K* balance instances set and used this set to do the decision tree learning process.

If we only chose *K* from the negative class instances, we would miss lots of information which belongs to the rest of negative class instances. Instead of randomly choosing *K* instances from negative class, we equally divided the whole negative class instances into several sections, and each section contains *K* instances. Let's say we had N sections; we put the *K* positive instances into each section, thus, we had new *N* sections which contained the same positive class instances, but different negative class instances. For each section, we applied the decision tree learning. After *N* times decision tree learning, we had *N* different trees. Figure 3.6 shows how to generate multiple trees. Once we wanted to predict the class label for a new instance, we used these *N* trees independently to do the classification and returned the majority class to the new instance.

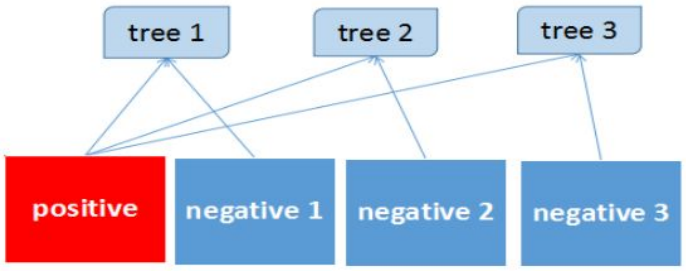


Figure 3.6 — The red block represents all positive instances, and each blue block represents one section of negative instances.

3.3 Random Forest

3.3.1 Motivation of using Random Forest

Random Forest is a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest [13]. In standard trees, a feature will be selected to be the internal node as long as this feature obtains a best split to decrease the entropy of the sample space. In Random Forest, each node will choose the best feature among a subset of the whole features set. Instead of generating a single tree in decision tree algorithm, Random Forest will generate a large number of trees, and these trees will vote for the most popular class. Therefore, we have two parameters, size of feature subset for each node and the number of trees. Actually, these two parameters are easily controlled and implemented based on the decision tree algorithm.

Each single tree is weaker than a standard tree, but the combination of all the trees is powerful. We can say that each single tree is like a specialist in a small field. Once the new instance goes through all the trees, it is similar to analyzing the instance in different aspects.

3.3.2 Random Forest with Imbalance data

As mentioned before, we generated multiple trees to solve the imbalance data set problem. According to Random Forest, a large number of trees were created, so we wanted to use these large number of trees to solve imbalance data problem.

We did not change the algorithm of Random Forest, but we changed the data set for Random Forest.

Algorithm 1: Random Forest with imbalance data set

Input: training set
Output: Random Forest model
Constant: Z represents the number of trees in Random Forest; K represents the number of instances in positive classes
Variables: i represents the *i*-th tree;
1: set a number for Z
2: find the K positive instance in training set
3: **for** i=1 to i=Z **do**
4: randomly choose K negative instances
5: mix the K positive instances and K negative instances
6: using 2K mixed instances as data set, and implementing the subset-feature strategy to build decision tree
7: **end for**

Random Forest based on the same data set to generate multiple trees. In step 2, we used the same *K* positive instances (because they were the minority), but used the different *K* negative instances at different times. Therefore, the only difference between any two new data sets was the *K* negative instances. All the negative instances have the similar properties, because they represented the non-behavior actions. Thus, we assumed this trivial change might not influence the Random Forest algorithm.

3.3.3 Selecting a combination of forest size and subset size

As mentioned above, there are two parameters in Random Forest: setting a subset-feature for each node and generating a large number of trees. According to Paper [13], they

implemented 100 trees, and the size of subset-feature is $Integer(\log_2^M + 1)$ which M is the size of the entire size. But we found that, using these two parameters did not obtain a significant improvement in our data set. Therefore we wanted to find the best combination of the subset-feature size and the forest size.

We found these two parameters by using the validation set. First, we set a matrix where the rows were the possible forest sizes, and the columns were the feature subset sizes. Then, we tried each combination to build Random Forest, and used the validation set to test the performance of Random Forest. After running over all the combinations, the best combination was determined by the best performance of the validation set. Thus, using the best combination of subset-feature size and forest size to build Random Forest.

3.4 Threshold Selection with Validation Set

After Random Forest was built, we got the initial classifications of each instance in the test set. However, according to the experiment, we found that using the majority voting strategy was not always correct. For example, when there was a negative class instance, Random Forest still had 55% trees predicted this instance as a positive class; only 45% trees predicted this instance as a negative class. Thus the instance should be classified as positive in terms of the majority voting strategy. There were a lot of incorrect classifications happening this way. Therefore, we wanted to find a threshold to determine the class for each instance, not just use the majority voting method.

We found the threshold based on the validation set. We used Random Forest to classify instances. In the validation set, for each instance, we could get the following information: how many trees predict this particular instance as slapping class, drinking class, or no action class (the data set only contains these three classes). We use *S*, *D* and *N* to represented those number of trees respectively.

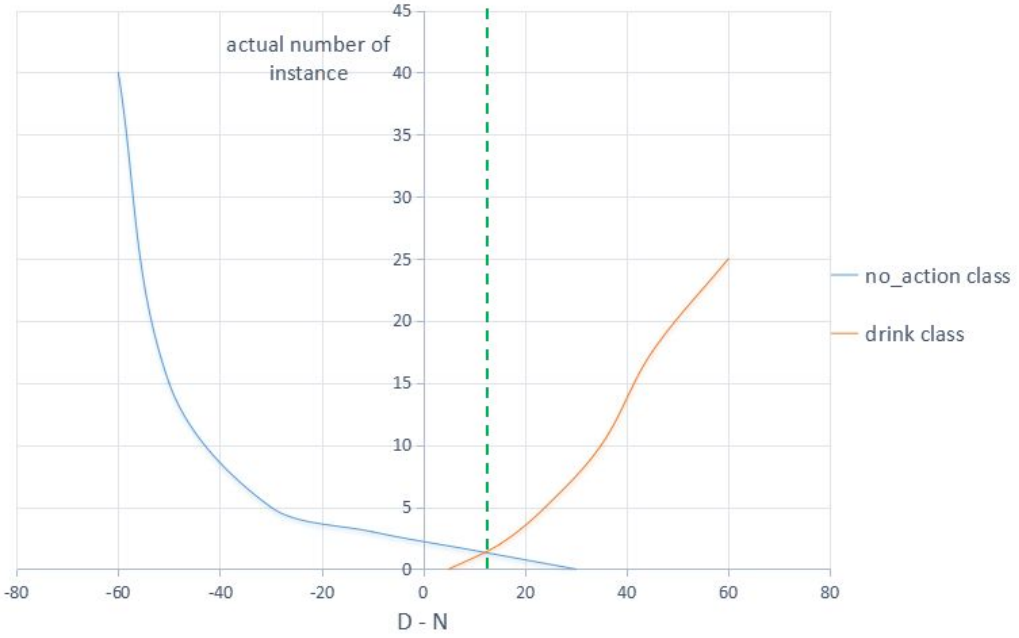


Figure 3.7 — The green dash line is the threshold

Figure 3.7 is an example of choosing the threshold for the drinking class. The X axis represents the number *D* - *N*. The Y axis represents the number of instances. The red curve represents the instances in the drinking class. The blue curve represents the instances in the no-action class. As we explained before, using Random Forest we can get *S, D* and *N* for each instance. If an instance shows strong characteristics of being a drinking class, then the *S* and *N* will be 0, and *D* will be the number of all the trees in Random Forest. If we assume there are total *R* trees in Random Forest, then the range of *D* - *N* is [-*R* , *R*]. Since we already know the class of each instance, the red and blue curves will be plotted. We expect the blue curve will decrease rapidly at first and then slowly decrease until it reaches zero, while the red curve slowly increases. In Figure 3.7, there is a cross point of the two curves, so the X axis value of the cross point will be the threshold for *D* - *N*. For a testing instance, *S, D* and *N* are generated after computing Random Forest. Once *D*-*N* is smaller than the threshold, the instance will be labeled as the no action class; otherwise the instance will be labeled as the drinking class.

The threshold for the slapping class will be generated in the same way, but the X axis will be changed from *D - N* to *S - N*.

3.5 Sequence Post-Processing

The Sequence Post-Processing is to potentially correct some missed classifications by a tree or Random Forest.

Here is an example to explain the motivation of using this method: We assumed that a drinking action lasts two seconds. As we know, each instance contained four continuous windows and each window contained one-quarter second records. Thus, eight continuous drinking instances would represent this two-second drinking action. So if only one drinking instance existed while the forward and backward instances belonged to the no action class, this drinking instance must be a wrong classification.

In fact, there were several situations we needed to analyze; the example above is just one. So we wanted to find all the situations, and used the patterned information to improve our performance.

Algorithm 2: Sequence Post-Processing

Input: the training set; the classifications of all the instances in the test set; the S, D and N for each instance
Output: the new classifications of all the instances in the test set.
Constants: drink_gap represents the minimum gap between one drink action and another action; drink_min and drink_max represent the minimum length and maximum length among all the drink actions.
Variables: drink_remain represents the number of instances which are contained in the previous drink action; continuous_length represents the number of instances in one action.

```
1: Using training set to find these constants: drink_gap, drink_min, drink_max
2: initially sign drink_remain = 0
3:for all the instances in the test set do
4:    if i-th instance is Drinking class
5:        if drink_remain > 0
6:            count the continuous_length of drinking class
7:            drink_remain=drink_remain+continuous_length
8:            if drink_remain > drink_max
9:                for i=drink_max to i=drink_max+drink_gap do
10:                   set i-th instance as No_action class
11:               end for
12:               drink_remain=0;
```

```
13:             end if
14:         end if
15:         else if drink_remain = = 0
16:             count the continuous_length of drinking
17:             if continuous_length < drink_min
18:                 increase continuous_length to drink_min
19:                 drink_remain = drink_min;
20:             end if
21:             else if continuous_length > drink_max
22:                 for i=drink_max to i=drink_max+drink_gap do
23:                     set i-th instance as No_action class
24:                 end for
25:                 drink_remain = 0
26:             end if
27:             else if drink_min <=continuous_length<=drink_max
28:                 drink_remain = continuous_length
29:             end if
30:         end if
31:   end if
32:   if i-th instance is No_action class
33:         if drink_remain = = drink_max
34:               while i<current position + drink_gap do
35:                     set i-th instance as No_action class
36:               end while
37:               drink_remain = 0
38:         end if
39:         else if 0<drink_remain<drink_max
40:             count the continuous_length of No_action class
41:             if continuous_length >= drink_gap
42:                 drink_remain = 0
43:             end if
44:             else if continuous_length < drink_gap
45:                 change all the instance which belongs to continuous_length as Drinking class
46:                 drink_remain=drink_remain+continuous_length
47:                 if drink_remain > drink_max
48:                     for i=drink_max to i=drink_max+drink_gap do
49:                         set i-th instance as No_action class
50:                     end for
51:                 end if
52:             end if
53:         end if
54:   end if
55:end for
```

In line 1, we calculated drink_gap, drink_min, and drink_max from the training set. drink_gap represents the minimum number of instances in No_action class which are between one drink action and another action; drink_min and drink_max correspond to the

minimum and maximum number of instances among all the drink actions. Lines 4 to 31, showed what we would do if there was an instance in drinking class. The **if** statement from lines 5 to 14 showed what the program will do if a remained drink action. The **else if** statement from lines 15 to 30 showed what the program will do if no remained drink action. Moreover, once there was no remained drink action, we would have three situations which are: drink action is smaller than drink_min, larger than drink_max, or between drink_min and drink_max. Lines 32 to 54 showed what we would do if there was an instance in No_action class. The **if** statement from lines 33 to 38 ensures that there is enough space between two actions as long as the previous drink action reaches drink_max. The **else if** statement from 39 to 53 help us group multiple short drink actions into one long drink action.

The algorithm showing above considered these points:(1)in order to be a legal drinking action, the number of instances of drinking class must between drink_min to drink_max;(2) any two actions cannot be closer than drink_gap;(3)a long action can be split as multiple small actions.

This algorithm was not perfect, there was a compromise. We pursued a higher accuracy in drinking classifications, but lost the accuracy in No_action classifications. Furthermore, we did not involve the slapping class in this algorithm, because we already got an accurate classifications for slapping instances without using this algorithm. In fact, we actually had implemented the slapping class into this algorithm, but there was no improvement for the true positive rate of slapping. Until now, the Sequence Post-Processing still has in good influence in the overall performance.

Chapter 4
Experimental Evaluation

4.1 Experimental Data

In this paper, we used 3 data sets to evaluate our algorithm performance. All the data came from a wearable sensor, which was located on the user's wrist.

The first data set includes 9 knee-slapping actions. This data set has 47 seconds of data and each second has 33 records.

The second data set includes 10 head-slapping and 5 tea-drinking actions. These actions happen randomly. This data set has 1773 seconds of data and each second has 100 records.

The third data set includes 29 head-slapping and 29 tea-drinking actions. This data set has 310 seconds of data and each second has 100 records.

The head-slapping and knee-slapping motions last approximately 0.5 second, and the tea-drinking motion lasts around 2 seconds for all data sets mentioned above.

4.2 Evaluation Criteria

As mentioned before, the given data set is imbalanced: the accuracy of the overall test set cannot relate the performance of the system. We set the slapping and drinking motions as the different positive classes, and we set the rest of time ("no actions") to be the negative class. Therefore, we used the true positive rate and the false positive rate to be the criteria. We expected high true positive rates in both slapping- and drinking-class motions, and a low false positive rate in "no actions" class at the same time.

Table 4.1 — The matrix of classified classes numbers

		Predicted class		
		slapping	drinking	no action
Actual class	slapping	a	b	c
	drinking	d	e	f
	no action	g	h	i

According to Table 4.1, the true positive rate of slapping is a/(a+b+c), the true positive rate of drinking is e/(d+e+f), and the false positive rate of no action is (g+h)/(g+h+i).

The accuracy above is determined by each instance. However, in behavior analysis, we can also predict that whether the system can correctly identify each slapping action and drinking action. We will not use false positive rate to calculate the error rate, because we do not know how many no actions will be in the test set. Instead of using false positive rate, we will use precision rate, which is to evaluate the accuracy of the predicted classes, for both slapping and drinking classes.

According to Table 4.1, the recall rate of slapping is a/(a+b+c), the precision rate of slapping is a/(a+d+g); the recall rate of drinking is e/(d+e+f), the precision rate of drinking is e/(b+e+h).

4.3 Experimental Procedures

The raw data was recorded in Excel files; because these files were not the input files for our program, the first step was to change the Excel files into text files.

In the next step, we extracted the features from the raw data set. We set 25 records in one window, and we used the continuous 4 windows to be one instance. 39 features were extracted in one window; thus, 39 * 4= 156 features in one instance.

For machine-learning purposes, we divided the data set into a training set and a test set. We used 1/4 of the whole data set to be the test set, and 3/4 of the whole data set to be the training set. In the training set, we chose 1/3 data to be the validation set; the validation set was not involved in the training process. The training, testing, and validation sets kept in similar class distributions.

We compared the accuracy by implementing the different components of our system:

Table 4.2 — The different methods with different components

	Decision Tree	Random Forest	Sequence Post-Processing	Validation Threshold	Best Combination of Random Forest
method 1	√				
method 2		√			
method 3		√	√		
method 4		√	√	√	
method 5		√			√
method 6		√	√	√	√

In method 1, we wanted to see the performance by implementing only Decision Tree algorithm. In method 2, we wanted to compare the difference between Decision Tree and Random Forest. In method 3, we expected that there was some improvement while implementing Sequence Post-Processing on Random Forest. In method 4, we expected that adding validation threshold could help the system improve the accuracy. In method 5, we wanted to test whether using the best combination of forest size and subset-feature size could improve the performance of Random Forest. In method 6, we added all the components, and expected that it had the best performance among the six methods.

For each method, we completed ten experiments with different pairs of training and testing sets. Two true positive rates and one false negative rate were generated in each experiment, and we reported the average value of the ten experiments.

4.4 Experimental Results

The first data set contained 9 knee-slapping motions. The results are shown in Table 4.3 and Table 4.4.

Table 4.3 — True positive rate and false positive rate of the first data set

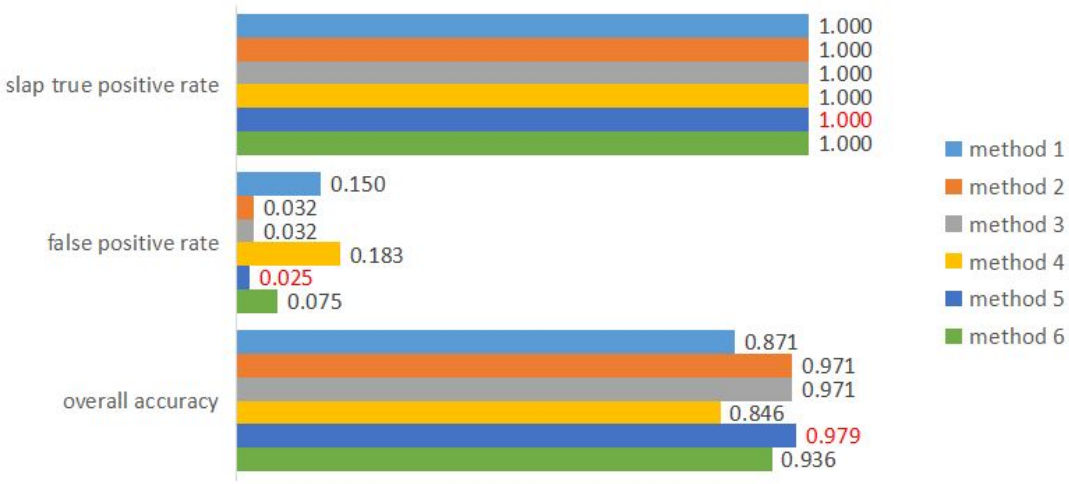
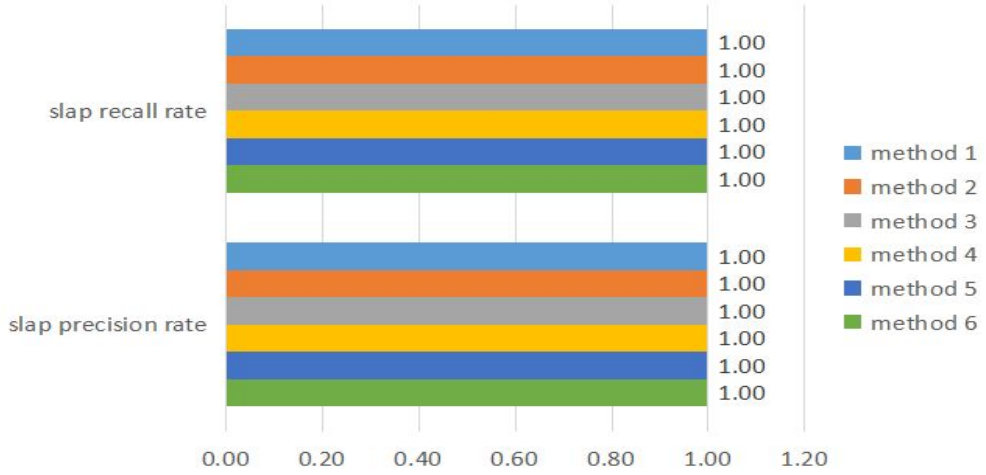


Table 4.4 — Actions recall rate and precision rate of the first data set



The running time for Decision Tree is 0.153 second, for Random Forest is 0.844 second.

We concluded the following points from Table 4.3: (1) The true positive rate of the slapping class was 100% in all methods. (2) Method 2 and 3 had the same result because Sequence Post-Processing did not affect the slapping class. (3) In method 4, we implemented the validation threshold, but the false positive rate increased, and the overall accuracy decreased. Because there were only 9 slapping actions in this data set, part of the 9 actions will be split to be the validation set, which makes the threshold less accurate. (4) In method 5, we used the combination of 100 trees and 15% subset-feature rate in Random Forest. Comparing with other methods, method 5 obtained the highest true positive rate in drinking class, and lowest false positive rate in No_action class, which proved that the combination of the two parameters improves the accuracy for Random Forest. (5) As we observed from method 4, implementing validation threshold affect the false positive rate. Method 6 contained all the components in method 4, thus the false positive rate in method 6 must be influenced by the validation threshold. However, the false positive rate of drinking in method 6 is lower than it was in method 4, which directly proved that the best

combination of two parameters for Random Forest can improve the accuracy of validation threshold.

For the recall rate and precision rate, all the methods have 100%. One possible reason is that the data set is less complicated and each slapping action can be easily identified.

The second data set contained 10 head-slapping and 5 tea-drinking motions. The results are shown in Table 4.5 and Table 4.6.

Table 4.5 — True positive rate and false positive rate of the second data set

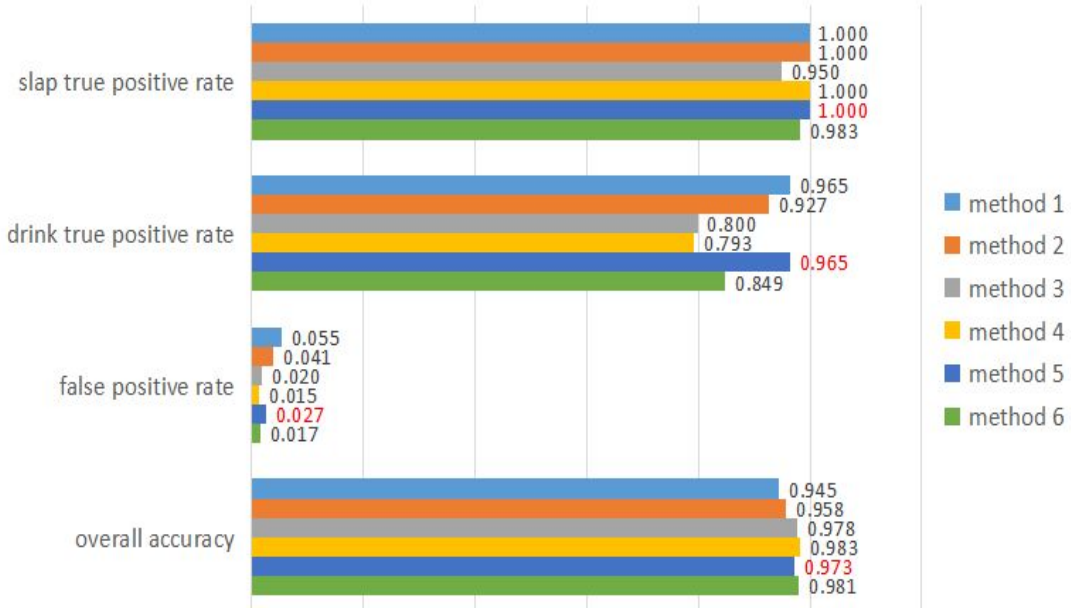
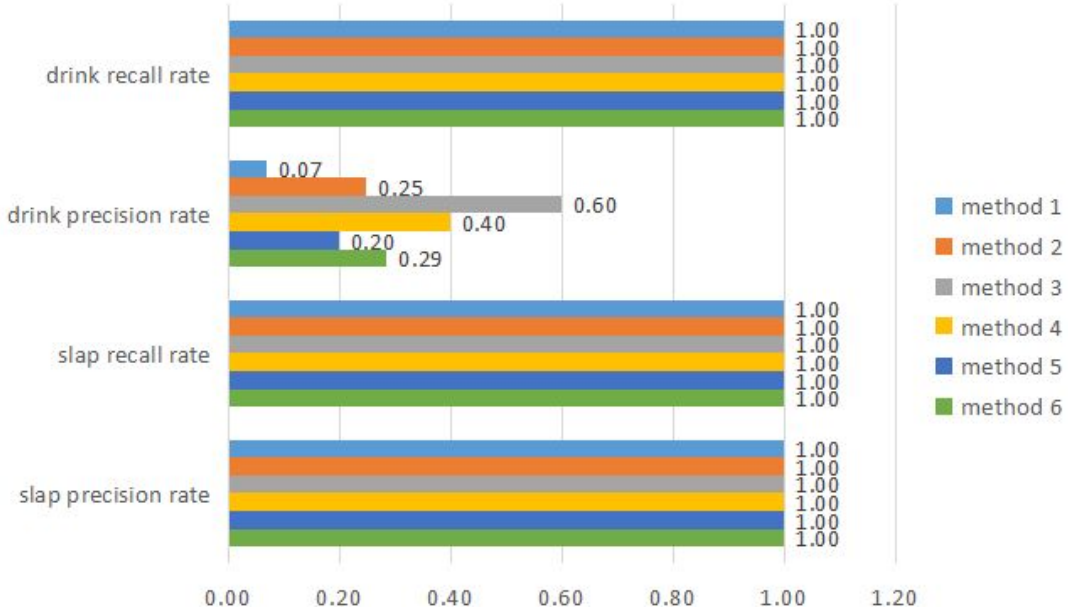


Table 4.6 — Actions recall rate and precision rate of the second data set



The running time for Decision Tree is 3.963 second, for Random Forest is 2.852 second.

According to the results, the slapping class still gave a good true positive rate. In method 1, using the normal standard Decision Tree could already enable us to obtain high accuracy in both slapping and drinking classes. In method 2, Random Forest decreased the false positive rate of no action, but also decreased the true positive rate of drink. In method 3 and method 4, we obtained the low false positive rates: 2% and 1.5%; however, the true positive rates of drinking class significantly fell to 80% and 79.3%. The trade-off of drinking or no action existed in method 3 and 4. As the results indicated in method 5, finding the best combination of forest size and subset-feature size for Random Forest is an expected and reasonable method. We implemented 100 trees and 20% subset-feature rate for Random Forest in this data set. In method 6, since we implemented the best combination of two parameters for Random Forest, the true positive rate of drinking became better than it was in methods 3 and 4, but was still lower than it was in methods 1, 2 and 5. However, the false positive rate in method 6 is much lower than it was in method

1 and 2. Therefore, we could conclude that Sequence Post-Processing and validation threshold had the tradeoff of the true positive rate in drinking or the false positive rate in No_action in this data set. Considering the overall performance, we thought that method 5 was the best choice.

In the second data set, there are totally five drink actions, some of them are located in the test set. Therefore the actual drink action in test set will be one or two, as long as there are some incorrect classified drinking actions, it will rapidly decrease the precision rate of drinking. That is one possible reason that in all methods the precision rate of drinking is low. However, in method 3 after Sequence Post-processing implemented in the system, it prevent predicting drink actions that are too short. That is one possible reason to increase the precision rate of drink action in method 3.

The third data set contained 29 head-slapping and 29 tea-drinking motions. The results are shown in Table 4.7 and Table 4.8.

Table 4.7 — True positive rate and false positive rate of the third data set

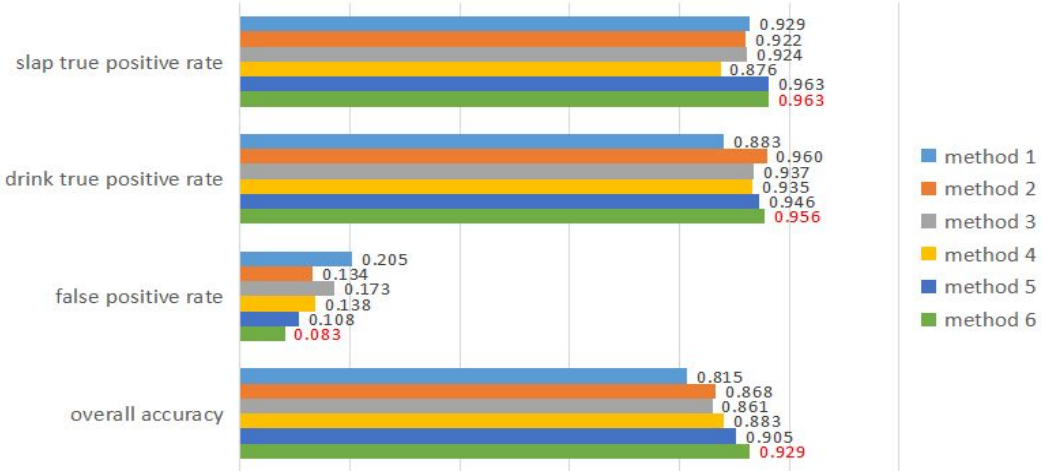
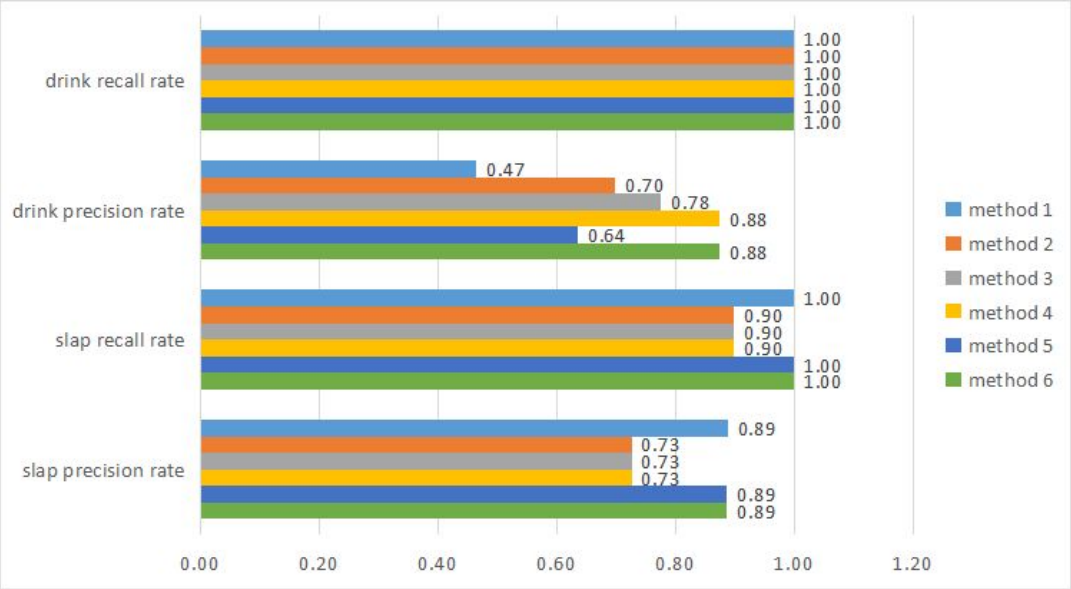


Table 4.8 — Actions recall rate and precision rate of the third data set



The running time for Decision Tree is 5.789 second, for Random Forest is 80.264 second

As we could see from Table 4.5, we could not obtain a 100% accuracy by using Decision Tree. Moreover, the accuracy of drinking class fell to 88.3%, and the false positive rate raised to 20.5%. According to the results from Decision Tree, the data set might contain noise. After implementing Random Forest in method 2, the accuracy of drinking increased, and the false positive rate fell to 11.7%. These results indicated that Random Forest had a strong ability to handle the noisy data. From method 3 to method 4, there was no improvement by implementing the validation threshold or Sequence Post-Processing. On the contrary, using validation threshold in method 4 had a bad effect on the accuracy of slapping. One possible reason was that Random Forest was weak in finding the threshold of slapping class. In method 5, we implemented 100 trees and 30% subset-feature rate for Random Forest. As we expected, once implementing the best combination of forest size and subset-features size of Random Forest, the performance significantly improved. The accuracy of slapping and drinking reached 96.3% and 94.6% respectively, and the false

positive rate decreased from 13.8% to 10.8%. In method 6, we surprisingly found that the false positive rate fell to 8.3%, the true positive rate of drinking increased from 94.6% to 95.6%, and the true positive rate of slapping remained the same. It proved that using validation threshold and Sequence Post-Processing had a good effect on decreasing the false positive rate of No_action, and increasing the true positive rate of drinking in this data set.

By only using Decision Tree, the drink precision rate just 47%, which means there are lot of predicted drink actions are incorrect. However, adding Random Forest the drink precision rate increased to 70%, it indicated that Random Forest has a better performance than Decision Tree. After we implemented validation threshold and Sequence Post-processing, the drink precision rate reach at 88%. As we expected, after implemented the method 6, in both recall rate and precision rate we can obtain the best performances. It proved that some illegal actions can be eliminated by Sequence Post-processing. As the result shown in third data set, finding the best combination of forest size and size of feature subset for Random Forest is important.

Chapter 5
Conclusions

5.1 Contributions

According to the experimental results, we obtained an over 96% true positive rate of the slapping class in the three data sets. At the same time, we also obtained an over 95% true positive rate of drinking class. These achievements gave us confidence in detecting the harmful hand actions and distinguishing harmful hand actions from normal hand actions.

Secondly, we implemented the best combination of forest size and subset-feature size for Random Forest, which gave a better performance rather than just use the figures given by Paper [13]. For different data sets, the combination of these two parameters may change, so finding these two parameters for different data sets was necessary.

Thirdly, we designed the Sequence Post-Processing method to correct the missed classifications which came from Random Forest. We found the pattern information from the training set and used this information to build the model. As the results show above, for the noisy data set, the Sequence Post-Processing did help the system decrease the false positive rate.

5.2 Limitations and Potential Improvement

Sequence Post-Processing processed the instances group by group. However, in behavior analysis, analyzing the instance one by one is probably better. We prefer the system recognize the behavior in the current time rather than recognize the behavior after a certain period of time. In the future, we will try to analyze the instance one by one by using the pattern information from training set.

Efficient features are always the important part in classification problems. There were at least two aspects we can optimize the features. The first aspect is to extract more types of features. Right now the data sets only contain three different classes, we can get a good performance on the current data sets, but we cannot guarantee these features are efficient if the type of class increases. The second aspect is to decrease the feature dimensions, such as using PCA to decrease the feature dimensions.

In the third data set, there are several short time drinking actions. The system right now recognizes the multiple short-time drinking action as a long time drinking action. Even though it does not influence the true positive rate of drinking class, we expect to detect each single drinking action.

In the data pre-processing part, we manually found the starting time and ending time for each action, but once the data set gets much larger than before, this procedure is not available. Therefore, we may design a method which can use the raw data to find the starting and ending times for each action.

References

1. Krause, A., Siewiorek, D. P., Smailagic, A., & Farringdon, J. (2003, October). Unsupervised, dynamic identification of physiological and activity context in wearable computing. In null (p. 88). IEEE.

2. Ugulino, W., Cardador, D., Vega, K., Velloso, E., Milidiú, R., & Fuks, H. (2012). Wearable computing: accelerometers' data classification of body postures and movements. In Advances in Artificial Intelligence-SBIA 2012 (pp. 52-61). Springer Berlin Heidelberg.

3. Mannini, A., & Sabatini, A. M. (2010). Machine learning methods for classifying human physical activity from on-body accelerometers. Sensors, 10(2), 1154-1175.

4. Li, C., Lin, M., Yang, L. T., & Ding, C. (2014). Integrating the enriched feature with machine learning algorithms for human movement and fall detection. The Journal of Supercomputing, 67(3), 854-865.

5. Yang, J. Y., Wang, J. S., & Chen, Y. P. (2008). Using acceleration measurements for activity recognition: An effective learning algorithm for constructing neural classifiers. Pattern recognition letters, 29(16), 2213-2220.

6. Yang, X., Dinh, A., & Chen, L. (2010, April). Implementation of a wearerable real-time system for physical activity recognition based on naive Bayes classifier. In Bioinformatics and Biomedical Technology (ICBBT), 2010 International Conference on (pp. 101-105). IEEE.

7. Nam, Y., Rho, S., & Lee, C. (2013). Physical activity recognition using multiple sensors embedded in a wearable device. ACM Transactions on Embedded Computing Systems (TECS), 12(2), 26.

8. Hong, P., Turk, M., & Huang, T. S. (2000). Behavior modeling and recognition using finite state machines. In Automatic face and behavior recognition, 2000. Proceedings. fourth IEEE international conference on (pp. 410-415). IEEE.

9. Pylvänäinen, T. (2005). Accelerometer based behavior recognition using continuous HMMs. In Pattern Recognition and Image Analysis (pp. 639-646). Springer Berlin Heidelberg.

10. Wu, J., Pan, G., Zhang, D., Qi, G., & Li, S. (2009). Behavior recognition with a 3-d accelerometer. In Ubiquitous intelligence and computing (pp. 25-38). Springer Berlin Heidelberg.

11. Ward, J. A., Lukowicz, P., Troster, G., & Starner, T. E. (2006). Activity recognition of assembly tasks using body-worn microphones and accelerometers. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 28(10), 1553-1567.

12. Anjo, M. D. S., Pizzolato, E. B., & Feuerstack, S. (2012, November). A real-time system to recognize static behaviors of Brazilian sign language (libras) alphabet using Kinect. In Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems (pp. 259-268). Brazilian Computer Society.

13. Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.