Compression-based Activity Classification and Motif Discovery in Time Series of Acceleration Data

INF/SCR-08-04

Master Thesis
Jefrey Lijffijt
Applied Computing Science
Utrecht University

Supervisors:
Ingrid Flinsenberg, Philips Research
Arno Siebes, Utrecht University

September 22, 2008

Abstract

There is an increasing need for automated systems for health monitoring and care assistance. According to UN forecasts the relative number of elderly people in the world will be doubled by 2050, compared to 2005. Now people must visit a doctor to assess their ability to perform activities of daily living (ADLs), and discover diseases such as dementia. Systems can assist in this process by keeping track of the daily behavior of people. Such monitoring system are potentially more accurate, less invasive and require less time from medical experts than occasional personal doctor visits. To realize continuous monitoring systems, algorithms for activity recognition are required. In this thesis we present a new algorithm for recognizing activities based on a single wearable sensor, a triaxial accelerometer.

First, using motif discovery techniques, we show many activities such as walking contain strongly repetitive patterns and we show that such patterns are often similar for different people. Secondly we introduce a discrete representation for the numerical time series of acceleration samples and use the MDL-based data mining algorithm *Krimp* to build a classifier for activities on top of this representation. We show for activities containing basic motion patterns our new approach outperforms earlier algorithms. We do so by evaluating our algorithm on a publicly available data set.

Contents

Chapter 1

Introduction

We perform many activities of daily life in a similar fashion every day. We get up, have breakfast, go to work, come home again, have dinner, relax a bit and go to bed to repeat the same pattern the next day. It is possible to use pattern recognition algorithms on sensor data, from cameras or motion detectors, to monitor this daily rhythm.

Monitoring physical activity and it's daily rhythm is interesting for several reasons. For example Hayes et al. (2004) and Verghese et al. (2002) have shown dementia can be detected at an early stage by studying the variation in movement and gait of elderly. Monitoring activity patterns for abrupt changes can be used to detect falling, after which an emergency service or caretaker can be notified automatically as shown by Chen et al. (2005). Jain et al. (2006) have build a system to assist people with a memory impairment in daily life, which requires the computer system to be aware of 'normal' behavior. This list of applications in the medical domain is by no means exhaustive and the application also covers other domains. On the border of the lifestyle and medical domain, Consolvo et al. (2008) study motivating people to stay (or get) in shape by giving feedback on their energy expenditure. The energy expenditure estimate is based on activity tracking. In a more broad perspective activity recognition algorithms give operational context for programs, which will enable ambient intelligent systems as described in Verhaegh et al. (2003).

We focus on wearable devices for the recognition of activities. This topic is not new. For example Patterson et al. (2005), Wang et al. (2007) and Stikic et al. (2008) use a wrist-worn RFID reader to monitor the objects used by an individual, which in turn is used to predict the activity the person is involved in. The activities recognized are instrumental activities of daily living such as preparing a meal, taking medication, cleaning the home, shaving and brushing teeth. Another approach is using sensors to determine a person's location (in the home) and deduce activities such as dining or shopping as shown by Liao et al. (2005) or predict the path through a home as shown by Aipperspach et al. (2006) respectively. An interesting and perhaps less expected method is to use an on-body accelerometer to deduce what activities the user is engaged in. Among others, Bao & Intille (2004) and Huynh & Schiele (2006) have shown these movement patterns can be used to classify basic activities such as walking, running or sitting. Some studies use accelerometers along with other sensors, such as a heart rate monitor (Tapia et al. 2007), RFID tags

(Stikic et al. 2008), an array of sensors including temperature, infrared, pressure, microphone etc. (Van Laerhoven & Cakmakci 2000), or a house filled with wired and wireless sensors (Logan et al. 2007) to recognize all of the aforementioned activities. In Minnen et al. (2005) basic repetitive motions in activities are identified, which may be used for activity classification. At this moment there are small special purpose devices with accelerometers on the market which can be worn on a belt or in a trouser pocket. It is also possible to built accelerometers into watches or mobile phones. For example the Nokia 5500 Sports mobile phone is already equipped with such a sensor (Kirkeby & Kahari 2007).

In this thesis we present a new approach to building an activity classifier using a single triaxial accelerometer. In earlier studies on activity recognition, either an unreasonable amount or combination of sensors is used, or only some standard classification algorithms, such as decision trees, are used to build the classifier. We show activities can be deduced more robustly from this single sensor than has been shown so far. To do this we use a new algorithm which is better suited to handle this type of data.

This thesis is outlined as follows. We start with an introduction to the publicly available data set we use to train and evaluate the classifier in Chapter 2. In Chapter 3 we review motif discovery to find frequent patterns in activities. Based on the results we introduce a discrete representation for the data in Chapter 4. Also in Chapter 4 we discuss the compression-based classification algorithm *Krimp*, which uses frequent patterns similar to the ones found in Chapter 3, to describe activities. In Chapter 5 we present the results of the classification algorithm and use intermediate results of the algorithm to identify anomalous behavior. Finally, in Chapter 6 we summarize the main conclusions of this thesis.

Chapter 2

Data Set

The data set we use was produced by Ling Bao and Steven Intille as part of the House_n project (House_n Research Group 2007). One of the goals of the project was activity recognition for health care. The data set is publicly available at `http://architecture.mit.edu/house_n/data/`. This chapter is outlined as follows. In Section 2.1 we give an introduction to the data set, and in Section 2.2 we review the results of the associated paper from Bao & Intille (2004) to set the benchmark for Chapter 5. Subsequently, in Section 2.3 we discuss some initial preprocessing of the data, and in Section 2.4 we present an overview of the activities in the dataset based on their intensity. Finally in Section 2.5 we show how to clean up the data set.

2.1 General Information

For the ease of the reader we summarize the important properties of the data collection process as discussed in Bao & Intille (2004). The data set consists of samples from five biaxial accelerometers worn simultaneously on different parts of the body: hip, wrist, arm, ankle and thigh. The accelerometers were attached to the subject using medical gauze and all have a fixed orientation. One axis measures vertical acceleration, the other measures the forward acceleration. Technical specifications of the sensors can be found in Bao & Intille's (2004) article. Each of the accelerometers is sampling at 76.25 Hz, which is sufficient compared to the minimum of 20 Hz investigated by Bouten et al. (1997). Some derived properties of the data will be discussed throughout this thesis.

Twenty subjects, who were not affiliated with the researchers participated in the experiment. They received a gift certificate for ice cream as a reward for participating. The subjects are aged in the range 17 to 48, 13 were male and 7 female. For each of the subjects the experiment consisted of two parts. In the first part they had to do an obstacle course, in which they were handed out a list of assignments they could do in their own fashion. To minimize awareness of what was being measured assignments such as the following were given.

> *"Reward yourself by eating some snacks and drinking some water in the common room. Snacks, cups, and water from the sink can be found in the House_n kitchen."*

Subjects were given a watch and had to annotate start and stop times for activities themselves. For each subject between 82 and 160 minutes of activity was recorded. In the second part subjects had to do an activity course in which a list of activities was given which they had to execute in order. Because natural language definitions are often ambiguous, a description for each activity was given, see Bao's (2003) thesis. Again subjects had to annotate start and stop times themselves and no researcher or camera monitored them. For each subject between 54 and 131 minutes of activity was recorded in this fashion. Due to the weather some subjects skipped one or a few activities.

This data set is special for three reasons.

1. Real world circumstances.
   The accelerometers were not wired and could not be easily damaged so subjects could move around freely. Subjects were free to go outside of the laboratory. In the obstacle course subjects were even explicitly told to go outside.

2. Naturalistic and self-annotated by users.
   Between activities subjects had to write down start and end times and no one watched them execute their tasks.

3. Large number of daily activities.
   Many other data sets consist of only postures or specific movements or only a few daily activities. This data set includes sedentary, light, moderate and vigorous activities. Also some activities involve the entire body, while others are predominantly leg or arm based.

2.2 Classification Results

For building a classifier the open source package Weka (Witten & Frank 2005) was used. The algorithm giving the best result is C4.5 (Decision Tree) with an overall score of $84.26\% \pm 5.178$ (mean $\pm$ standard deviation) for leave-one-subject-out cross-validation. The other analyzed algorithms are Decision Table, Instance-Based Learning and Naïve Bayes. The general result of 84.26% was reached when using all five biaxial accelerometers as input for the classification algorithms. Since we use input from only one accelerometer in our algorithm, we summarize the Decision Tree results for experiments with one or two accelerometers in Figure 2.1. The extracted features were mean, energy, frequency-domain entropy and correlation using a sliding window of 6.7 seconds (512 samples) with 50% overlap. From each activity the first and last 10 seconds were removed because often this contained the activity of writing down the time while standing still, instead of the activity in the list.

Some basic knowledge can be extracted from the generated decision tree. For the sedentary activities (*sitting*, *standing still*, *lying down*) mean acceleration is the decisive factor. For ambulatory activities and *bicycling* the level of hip acceleration is very important. To distinguish between the high intensity activities *running* and *bicycling* the frequency domain entropy and arm-hip correlation are used. To discriminate between *working on a PC*, *eating & drinking*, *reading*, *strength training*, *window scrubbing*, *vacuum cleaning* and *brushing teeth* the arm posture, mean acceleration and energy are used. The activities

Accelerometer(s) Left In	Recognition Accuracy Difference
Hip	-34.12 ± 7.115
Wrist	-51.99 ± 12.194
Arm	-63.65 ± 13.143
Ankle	-37.08 ± 7.601
Thigh	-29.47 ± 4.855
Thigh and Wrist	-3.27 ± 1.062
Hip and Wrist	-4.78 ± 1.331

Figure 2.1: Difference in overall recognition accuracy for using one or two accelerometers instead of all five. Taken from Bao & Intille (2004).

stretching, *window scrubbing*, *riding the elevator* and *riding the escalator* have low recognition rates and require higher level analysis to be detected robustly.

In the analysis of the results the researchers conclude *"some activities are recognized well with subject-independent training data, others appear to require subject-specific training data"*. And thus *"there may be limitations to a pre-trained algorithm. Although activities such as* running *or* walking *may be accurately recognized, activities that are more dependent upon individual variation and the environment (e.g.* stretching*) may require person-specific training (Intille et al. 2004)"*.

In this thesis we build an algorithm for detecting the same activities while using only one sensor at the hip, because that is far more convenient to wear. The authors of the original paper already give a hint in this direction by stating *"Acceleration of the hip is the second best location for activity discrimination. This suggests that an accelerometer attached to a subject's cell phone, which is often placed at a fixed location such as on a belt clip, may enable recognition of certain activities."* It may be useful to combine classification results of hip and wrist to detect a bigger range of activities. Note the combination of the wrist with hip or thigh gives only slightly worse results compared to using all five accelerometers. We state one more conclusion from Bao & Intille's (2004) article.

> *"Classification accuracy rates between 80% to 95% for* walking, running, climbing stairs, standing still, sitting, lying down, working on a computer, bicycling *and* vacuuming *are comparable with recognition results using laboratory data from previous works."*

This statement will be our benchmark. We will come back to this in Chapter 5. As far as we know this data set has not been used in other papers yet.

2.3 Preprocessing

In the data set samples are taken from a biaxial accelerometer. In the rest of this section we assume samples are taken from a triaxial accelerometer. To adapt the following definitions from the general case to the data set we can simply ignore the z direction in the rest of this section.

Assume the input signal consists of samples taken consecutively from a triaxial accelerometer sensor. Each observation consists of the acceleration in m/s^2 in the x, y and z directions and a time stamp t in milliseconds. Since

the sample rate is approximately constant we omit the time stamp and simply number all samples $1, \ldots, n$. The entire sample set of n ordered observations from an experiment can be written as the series

$$T = (x_1, y_1, z_1); \ldots; (x_n, y_n, z_n) \quad \forall i \in \{1, \ldots, n\} : x_i, y_i, z_i \in \mathbb{R}. \tag{2.1}$$

The sensors also measure the gravity field, so if the device is stationary it will measure $1g$ acceleration. We assume we do not know the orientation of the x, y and z directions. If the orientation of the three axis would be constant we could calculate the absolute orientation in the three dimensional plane using earth's gravity. But because the sensor cannot be mounted exactly the same in each experiment and because the orientation can shift during an experiment (i.e. the device axes rotate) we cannot calculate the orientation accurately. This also means it is impossible to compute speed or displacement in any direction accurately. The orientation bias causes a serious problem when working with the raw sensor data. We explain this with a simple example.

Suppose our learning algorithm tries to find patterns in an experiment where our subject was walking. Assume the orientation of the sensor was very different from all other experiments. Because of the unique orientation of the sensor, the (combination of) values of the three axis might be unique as well. A supervised learning algorithm might learn this combination is indicative for walking. Since the unique values are caused by the orientation, the algorithm actually learned a pattern consisting purely of noise. This is of course disastrous for the generalization of the resulting models.

We can counter this problem easily by taking the L^2-norm of the input vector for each observation.

$$t_i = \sqrt{x_i^2 + y_i^2 + z_i^2} \quad \forall i \in \{1, \ldots, n\} \tag{2.2}$$

Along with reducing noise we get a free bonus of reducing the dimensionality from three to one. The new simplified list of n observations is given by

$$T = t_1, \ldots, t_n \quad \forall i \in \{1, \ldots, n\} : t_i \in \mathbb{R}. \tag{2.3}$$

This concludes our first preprocessing step. See Figure 2.2 for an example of preprocessed data. Next we show some basic differences between various activities.

2.4 Overview of Activities

The data set consists of twenty activities. All of the activities are considered to contain a specific basic motion. We only consider the hip sensor which brings a restriction on motions we can detect. In Figure 2.3 we find an overview of the relative intensity distribution for all activities in the data set, aggregated over all subjects. The acceleration for each activity is normalized to have mean zero as discussed in Chapter 4 and cleaned as discussed in Section 2.5. We immediately see some activities are alike while other activities can be easily distinguished from each other. The ambulatory activities *walking*, *carrying* and *climbing stairs* are very much alike, and very different from all others. *Running* has unmatched acceleration and is unique in it's distribution. *Sitting*, *working*

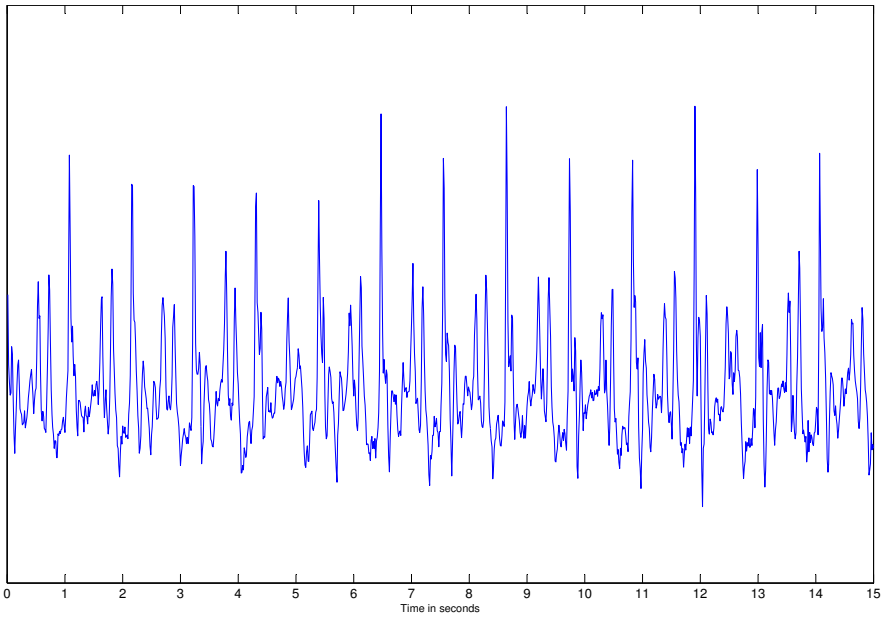

Figure 2.2: Preprocessed sample obtained from fifteen seconds of walking.

on PC, *watching TV*, *reading* and *lying down* contain virtually no acceleration at the hip sensor. *Standing still*, *eating/drinking*, *brushing teeth*, *folding laundry* and *riding escalator* contain only a very small amount of acceleration, which is probably just noise in the annotation. We expect it contains small parts of another activity, for example *walking*. The other activities contain enough acceleration to be detectable. Our challenge is to build an algorithm that classifies unseen data as one of the activities. We review the results in Chapter 5.

2.5 Cleaning Up

Annotation of the data has been done between activities by the participants themselves. Each part of the data set consists of a fairly long activity and often small parts of the data do not really belong to the activity. For example *standing still* appeared to contain parts of *walking*. In Bao & Intille (2004) it is argued this method of annotation is an advantage because hand-annotating data is costly and this approach enables collection of a larger data set at the same cost. While this is true, we can use an automated approach to clean up the data a bit.

Cleaning the data is necessary to ensure the estimated accuracy of the classifier reflects the true performance as well as possible. While training the algorithm noise is not a big problem. A learning algorithm should be robust to noise in the data set, and for our MDL-based algorithm this is especially true, because the algorithm captures only generalizations in the data. During

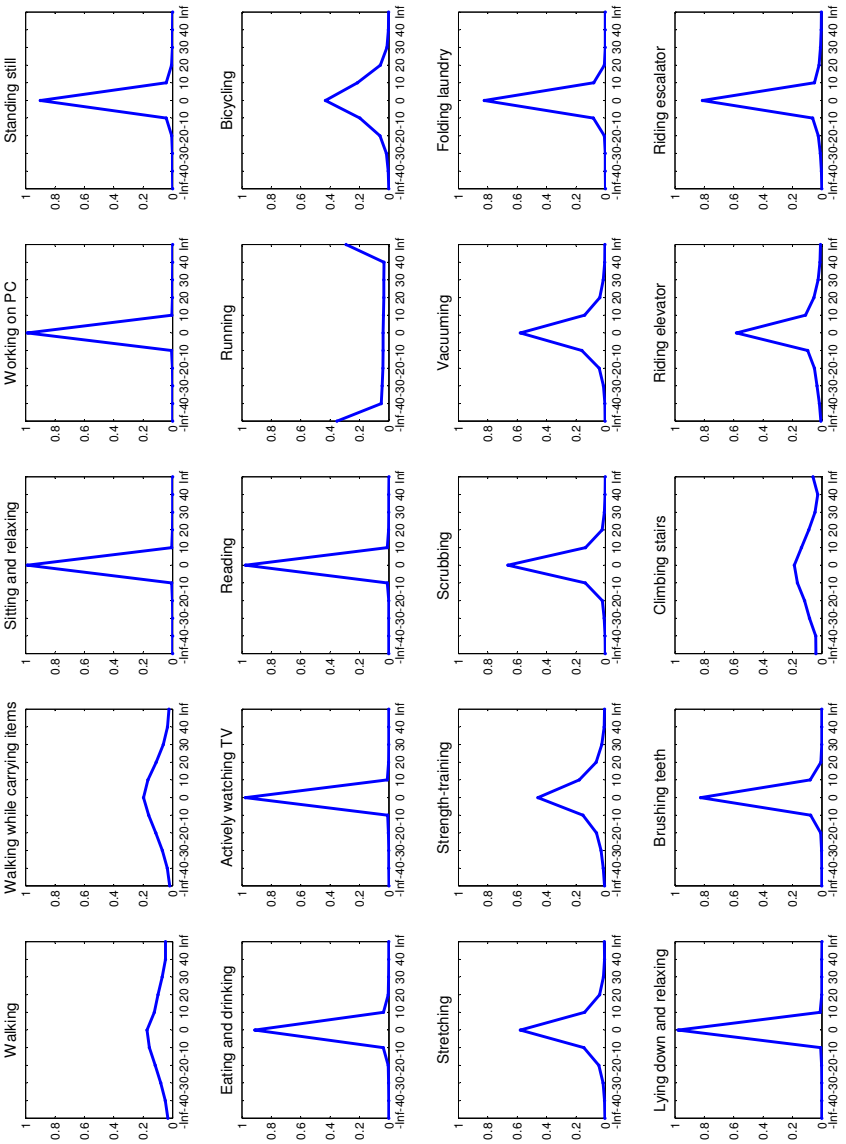

Figure 2.3: Histogram counts for all activities.

evaluation of the activity classifier noise can be a problem. To accurately measure the performance of the resulting classifier, the annotation error should be kept to a minimum. Suppose a test set contains data from the activity *sitting*, but is annotated with *walking* and suppose also our algorithm detects this as the activity *sitting* (which is not hard, as will be shown in Section 5). In our evaluation we measure an error, while in fact the recognition was correct. We use the following method for cleaning up the data from some of the activities.

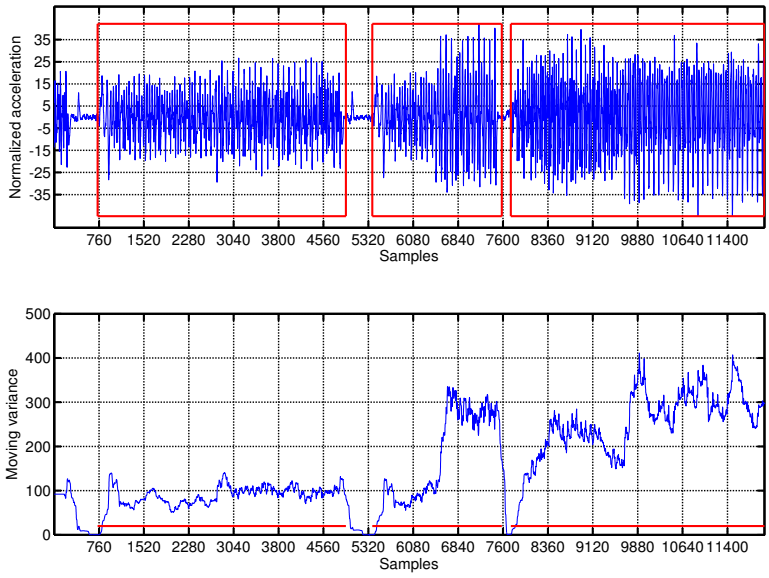

Figure 2.4: Selection of data from activity *walking* based on moving variance.

Consider Figure 2.4. The top shows the second sample of *walking* for the first subject, which clearly contains small parts of a passive activity. The bottom graph shows the moving variance over windows of the past 2.25 seconds ($2.25 * 76 = 171$ samples), except for the first 170 samples for which the window 1 to 171 is used. The selection indicated by the red line is where the variance is over 20. Because in Chapter 5 we use test sets equal to 10 seconds we select only the samples with variance over 20 for a consecutive period of 10 seconds or more. The moving variance has a response delay, to compensate for this the selection is shifted to the left by 1 second. The value 20 is chosen such that no data belonging to the class is thrown away. For example here the part with variance over 20 from the start is not selected. Based on the sample data selected in Figure 2.4 we conclude this method works very well for cleaning up the data. To give an overview how the cleaning procedure affects the full data set the following table lists all activities, the selection value, whether data should be above or below the selection value and the percentage of data selected.

Activity	Selection Value	Above/Below	Selection %
Walking	20	Above	93%
Carrying	20	Above	95%
Sitting	100	Below	100%
Working on PC	100	Below	100%
Standing	100	Below	72%
Eating/drinking	-	-	100%
Watching TV	100	Below	97%
Reading	100	Below	99%
Running	500	Above	70%
Bicycling	5	Above	90%
Stretching	5	Above	29%
Strength-training	5	Above	83%
Scrubbing	5	Above	55%
Vacuuming	-	-	100%
Folding laundry	-	-	100%
Lying down	100	Below	99%
Brushing teeth	-	-	100%
Climbing stairs	20	Above	90%
Riding elevator	-	-	100%
Riding escalator	-	-	100%

Selection for some activities, such as *stretching* and *scrubbing*, is quite strict, while for most activities almost all data is left as it is. In Figure 2.5 the selection of the second *stretching* sample for the third subject is plotted, which is typical for all samples of *stretching*. We find short acceleration patterns up to 5 seconds with no acceleration between these patterns. This may indicate a test set window of 10 seconds may be too short for this activity and also the moving variance window of 2.25 seconds may be too short for this activity to clean up the data. For most activities the moving variance window is sufficient and a short window makes it possible to detect short interruptions of an activity. Therefore we do not change the moving variance window length. *Stretching (arms and legs)* is probably very hard to detect anyway using a sensor mounted at a subject's hip. In the *stretching* data for subject three there is a fair amount of acceleration at the arm and wrist, but relatively scarce acceleration at the lower body.

In Figure 2.6 we find the selection of the first sample of *scrubbing* for the second subject. For *(window) scrubbing* also a lot of data is thrown away and also contains more acceleration at the wrist and arm than at the lower body. However, the acceleration between the lower and upper body is correlated and acceleration patterns do tend to last 10 seconds or longer. In the Figure we see most of the data not selected is flat, i.e. between -5 and 5. One could argue a moment of standing still can be part of the longer activity *scrubbing*, but because we search for basic motions in an activity we consider such a moment to interrupt the activity. Hence we expect the selection for *scrubbing* is appropriate.

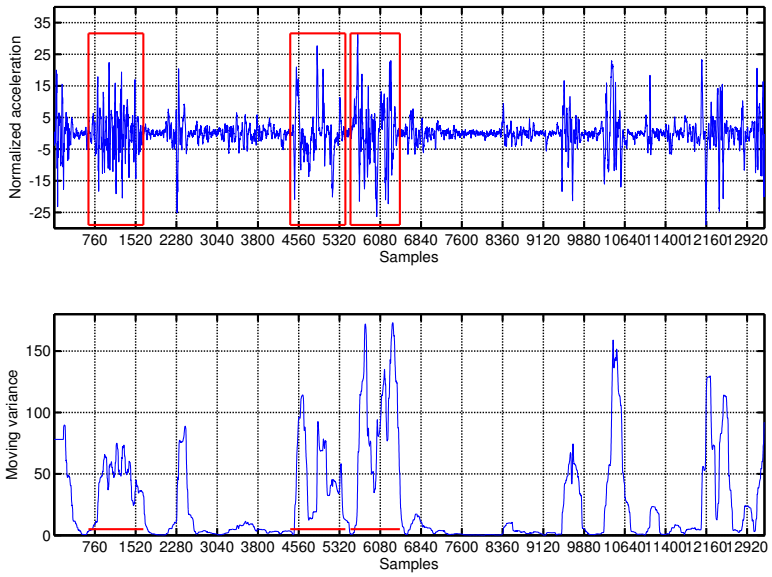

Figure 2.5: Selection of data from activity *stretching* based on moving variance.

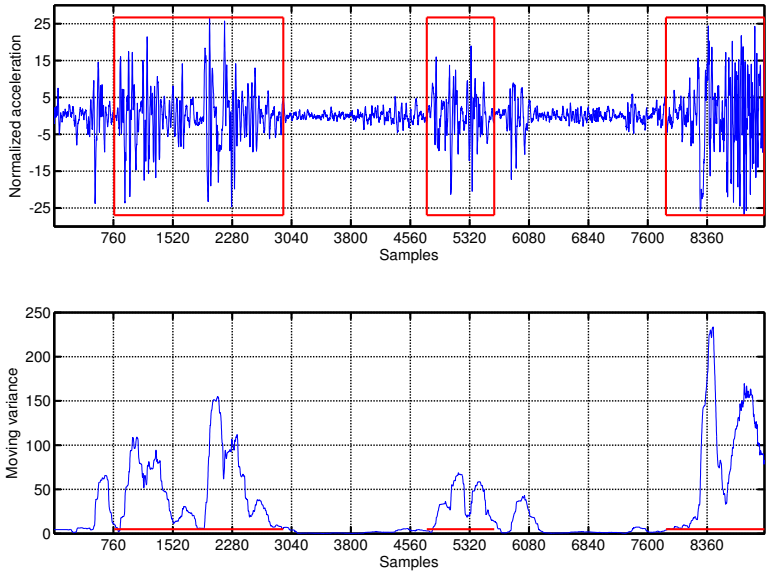

Figure 2.6: Selection of data from activity *scrubbing* based on moving variance.

Chapter 3

Motif & Discord Discovery

We expect certain activities, such as *walking* or *running* to have specific acceleration patterns. Our aim is to detect activities and abnormal situations, so in this chapter we discuss finding frequently recurring patterns and abnormal deviations. These two topics relate to several research domains in time series data mining: indexing, clustering and anomaly detection. Indexing is the problem of finding time series similar to an input series and clustering is the problem of finding natural grouping of similar time series. Anomaly detection is defined as searching subsequences of a time series deviating from a given "normal" situation. Searching for patterns in time series is also called *motif discovery*, hence we use the term *motif* to identify a frequently occurring pattern. Multiple approaches exist for finding *motifs* and in this chapter we discuss two algorithms. First, in Section 3.1 we discuss SAX, a fairly new algorithm, and in Section 3.2 we discuss the results of motif discovery using SAX. Secondly, in Section 3.3, we review the results using Euclidean distance to find motifs.

3.1 SAX

Symbolic Aggregate approXimation (SAX) is a symbolic representation for time series which can be used for many purposes (Lin et al. 2003). It is unique in the sense it is the only symbolic representation which allows lower bounding under the L^p-norm. It is relatively new but has already been shown a powerful and often used tool for time series analysis, see Keogh (2008). We use it here for motif discovery because it is simple and powerful and because the discrete representation used in the classification algorithm in Chapter 4 is inspired by SAX. Converting the input time series to the SAX representation is done in three steps. In this section we discuss each of them separately.

3.1.1 Sliding window

For motif discovery one usually cuts time series into smaller parts by using a sliding window. Instead of using the entire input series $T = t_1, \ldots, t_n$ algo-

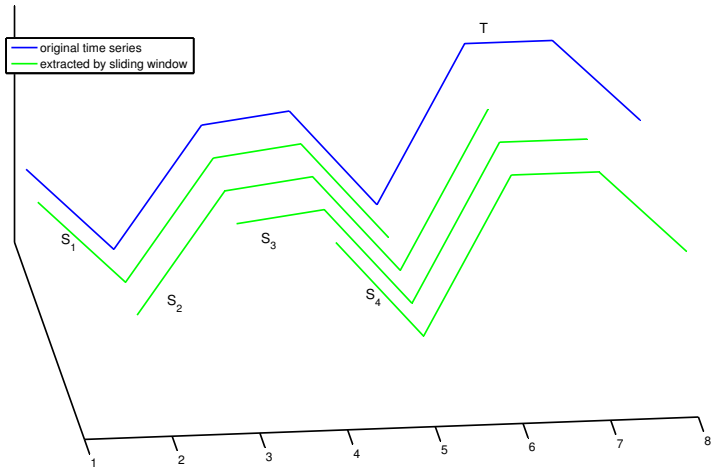

Figure 3.1: Result of cutting a sample with eight points into sequences of length five using a sliding window.

rithms work on the list of subsequences

$$\begin{bmatrix} S_1 & = \{t_1, \ldots, t_m\} \\ S_2 & = \{t_2, \ldots, t_{m+1}\} \\ & \vdots \\ S_{n-m+1} & = \{t_{n-m+1}, \ldots, t_n\} \end{bmatrix}.$$

Each subsequence has a fixed size m, the window length, which is given as input parameter to the motif discovery algorithm, in this case SAX. This parameter determines the length of the motifs we find. See Figure 3.1 for an illustration of the created subsequences. The strength of the sliding window lies mainly in using each observation as a starting point for a subsequence and normalizing each such subsequence independently. This ensures the dissimilarity caused by both time and value offset of the subsequences is minimized when comparing two sequences.

Normalizing each subsequence also causes values related to a single observation to differ per subsequences: $t_2 \in S_1 \neq t_2 \in S_2$. Since this is confusing we use the notation

$$S_i = \{s_{1,i}; \ldots; s_{m,i}\} \quad \forall i \in \{1, \ldots, n-m+1\}. \tag{3.1}$$

where the values of each subsequence S_i can be calculated by

$$s_{j,i} = \frac{t_{j+i-1}}{\sum_{k=i}^{m+i-1} t_k} \quad \forall j \in \{1, \ldots, m\}. \tag{3.2}$$

3.1.2 PAA

The following steps (PAA and discretization) operate on each of the subsequences separately. In the SAX method each subsequence S_i is normalized

independently to obey $\mu(S_i) = 0$ and $\sigma(S_i) = 1$, where μ is the mean and σ the standard deviation of all the elements in a subsequence. This minimizes the distortion caused by differences in offset, as illustrated in Figure 3.2. The second step in the process is to compute for each subsequence S_i the Piecewise Aggregate Approximation (PAA), denoted by $\overline{S}_i$, as $\overline{S}_i = \overline{s}_{1,i}; \ldots; \overline{s}_{w,i}$ with $w \leq m$, being the dimensionality of the approximation. Assume $\frac{m}{w} \in \mathbb{N}$. For any subsequence S_i we can compute the values $\overline{s}_{1,i}; \ldots; \overline{s}_{w,i}$ using the equation

$$\overline{s}_{j,i} = \frac{w}{m} \sum_{k=\frac{m}{w}(j-1)+1}^{\frac{m}{w}j} s_{k,i} \quad \forall j \in \{1, \ldots, w\}. \tag{3.3}$$

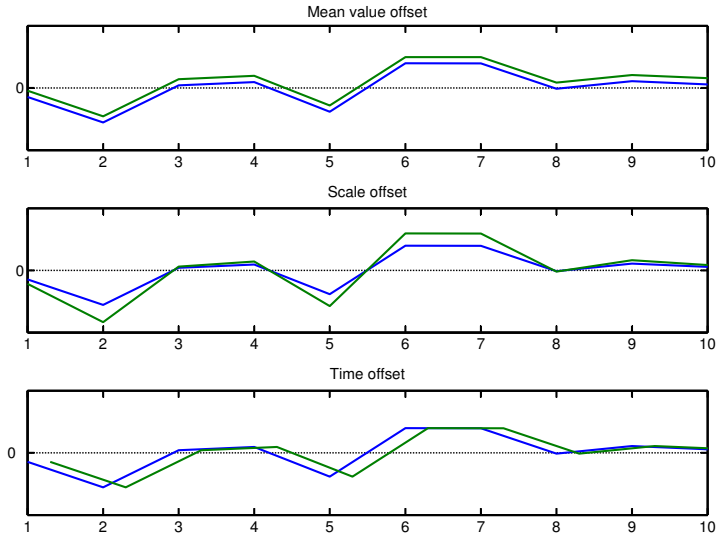

Figure 3.2: Subsequences that differ only by mean value (top), value scale (center) and time offset (bottom).

As said this reduces the dimensionality from m to w, which speeds up the motif discovery. PAA also smooths the input series because we average over multiple values. Although PAA can be defined for any positive integers m and w, we only choose m and w such that m divisible by w. We come back to choosing values for m and w at the experimental results in Section 3.2.

3.1.3 Discretization

The third and last step is discretization of the subsequence $\overline{S}_i$. In SAX, sample data is assumed to be normally distributed. In time series this is often the case and the assumption allows easy calculation of equiprobable intervals. As we can see in Figure 3.3, our data does not exactly fit a normal distribution, it

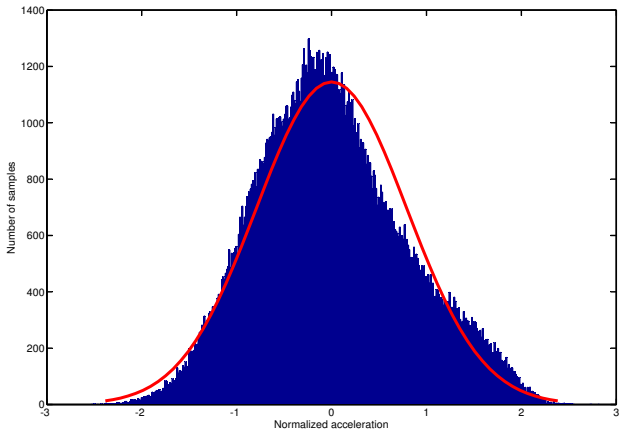

Figure 3.3: A histogram of the PAA values in the experiment in section 3.2 and a Gaussian distribution fitted to the values.

is a bit left skewed and has a higher peak. However, it sufficiently resembles the normal distribution for this step to work.

We ignore the difference and continue as if our data has a Gaussian distribution. Denote $\alpha = |A|$ the size of the alfabet $A = a_1, \ldots, a_\alpha$ we use. To discretize the data, we compute breakpoints $B = \beta_0, \ldots, \beta_\alpha$ dividing the input space into equiprobable regions. We define $\beta_0 = -\infty$ and $\beta_\alpha = \infty$. We select $\beta_1, \ldots, \beta_{\alpha-1}$ such that for any $\bar{s}_{j,i}$ we have $P(\bar{s}_{j,i} \in [\beta_0, \beta_1]) = P(\bar{s}_{j,i} \in [\beta_1, \beta_2]) = \ldots = P(\bar{s}_{j,i} \in [\beta_{a-1}, \beta_a])$. We can also derive the values from Table 3.1.

	$\alpha = 3$	$\alpha = 4$	$\alpha = 5$	$\alpha = 6$
β_1	-0.43	-0.67	-0.84	-0.97
β_2	0.43	0.00	-0.25	-0.43
β_3		0.67	0.25	0.00
β_4			0.84	0.43
β_4				0.97

Table 3.1: Lookup table for deriving breakpoints.

Assign all values in a certain interval to the corresponding character $\widehat{s}_{j,i} = a_k$ iff $\bar{s}_{j,i} \in [\beta_k, \beta_{k+1}]$ with $a_1 = a$, $a_2 = b$ etc. Now we end up with a series of characters $\widehat{S}_i = \widehat{s}_{1,i}; \ldots; \widehat{s}_{m,i}$ for each subsequence. We refer to this character series as the SAX *string* corresponding to a subsequence S_i. This concludes the conversion to the SAX representation.

3.1.4 SAX Summary

For the ease of the reader we summarize the steps once more, see Figure 3.4 for a graphical overview.

1. Cut the time series T into subsequences $S_1, \ldots, S_{n-m+1}$ using a sliding window and normalize each subsequence S_i to have $\mu(S_i) = 0$ and $\sigma(S_i) = 1$.
2. Compute for each subsequence S_i the Piecewise Aggregate Approximation $\overline{S}_i$.
3. For each subsequence discretize $\overline{S}_i$ to $\widehat{S}_i$ using the computed breakpoints $\beta_0, \ldots, \beta_\alpha$.

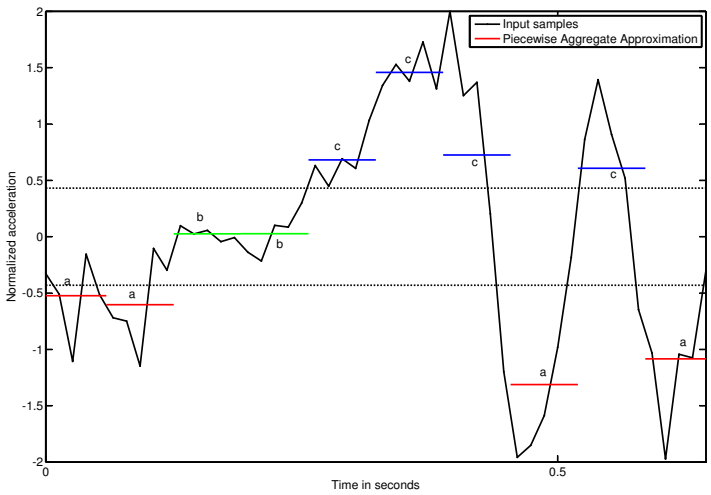

Figure 3.4: Computing the associated SAX string: $\{a, a, b, b, c, c, c, a, c, a\}$.

3.2 Motif discovery using SAX

For finding motifs with the SAX representation we simply count how many times each SAX string occurs. All strings with equal SAX strings are called matches.

Definition 1. Two SAX strings $\widehat{S}_i = \widehat{s}_{1,i}; \ldots; \widehat{s}_{w,i}$ and $\widehat{S}_j = \widehat{s}_{1,j}, \ldots, \widehat{s}_{w,j}$ are a *match* if and only if

$$\forall k \in \{1, \ldots, w\} : \widehat{s}_{k,i} = \widehat{s}_{k,j}$$

When using this approach with a sliding window there is a small problem we must address. The SAX string $\widehat{S}_i$ is very likely to be equal to the SAX string $\widehat{S}_{i+1}$. This is a trivial match we are not interested in. These trivial matches have caused a lot of trouble in previous time series research (Keogh & Lin 2005). We use the following definition inspired by Chiu et al. (2003) to define interesting matches.

Definition 2. A match between two SAX strings $\widehat{S}_i$ and $\widehat{S}_j$ associated with subsequences S_i and S_j respectively is *non-trivial* if and only if $|i - j| \geq m$.

There are other definitions possible. Such as defining the matching of two SAX strings is non-trivial iff there is a non-matching SAX string between the two subsequences. But the current definition suits our purpose. A maximum size set of non-trivial matches for a subsequence can be computed in linear time from the set of all matches. Algorithm 3.1 outputs a maximum size set. Note the solution may not be unique, but we compute only one solution. Given are a list of match indices I, ordered from low to high, and the window length m.

Algorithm 3.1 Compute largest set of non-trivial matches.

```
nonTrivial(I, m)
1  N = ∅
2  minIndex = 0
3  for i ← 1 to size(I)
4  do index ← I(i)
5     if index ≥ minIndex
6        then N.add(index)
7             minIndex ← index + m
8  return N
```

The parameters have to be set such that they produce motifs that are long but still have a lot of non-trivial matches. For this data set there is no test to measure the quality of a motif, so we have to judge this ourselves. We define the *best motif* as the motif having the largest set of non-trivial matches.

In the following experiments we use the data of the activity *walking* from the activity course (the second part of experiment) of subject 1. Consider Figures 3.5 and 3.6. Figure 3.5 shows the preprocessed sequence of all data annotated with *walking*, which covers approximately 241 seconds. The motif in the figure is the best motif found using a window length $m = 50$, a word size $w = 10$ and an alfabet size $\alpha = 3$. The rectangles drawn over the data indicate the positions of the non-trivial matches for the motif. Figure 3.6 shows the same sequence but is a zoom-in on 25 seconds of this data. The motif in the figure has a length of approximately 0.66 seconds, which is impressive for data from a process with high variation. In total it has 59 non-trivial matches. We can see in Figures 3.6 and 3.7 the subsequences belonging to the motif resemble each other very well. We conclude the sampled data for walking is quite stable. To validate this result we now look at motif discovery using Euclidean distance.

3.3 Motif discovery using Euclidean distance

We define the distance between two subsequences using the Euclidean distance (ED).

Definition 3. The *distance* between two subsequences S_i and S_j is defined as

$$d(S_i, S_j) = \sqrt{\sum_{k=1}^{m} (s_{k,i} - s_{k,j})^2}.$$

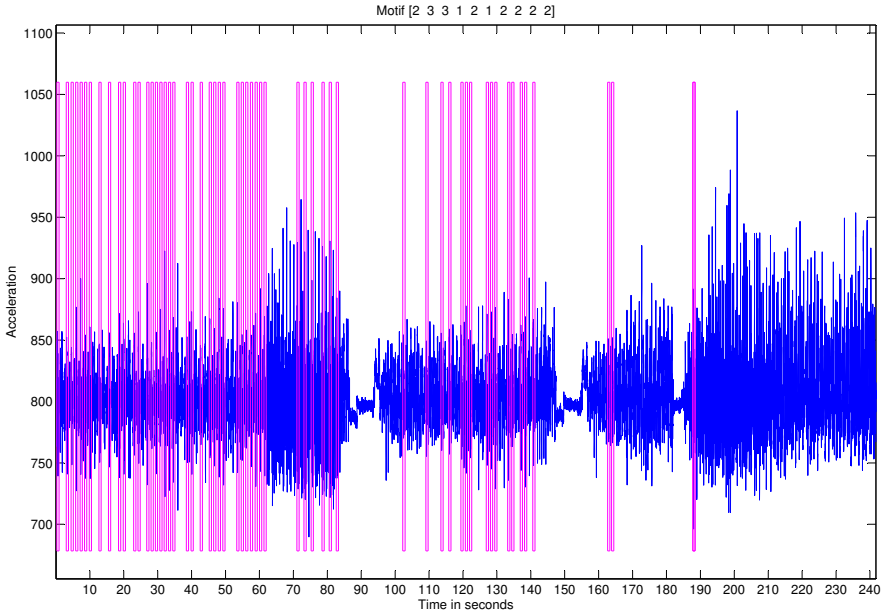

Figure 3.5: Overview of best SAX motif using parameters $m = 50, w = 10, \alpha = 3$.

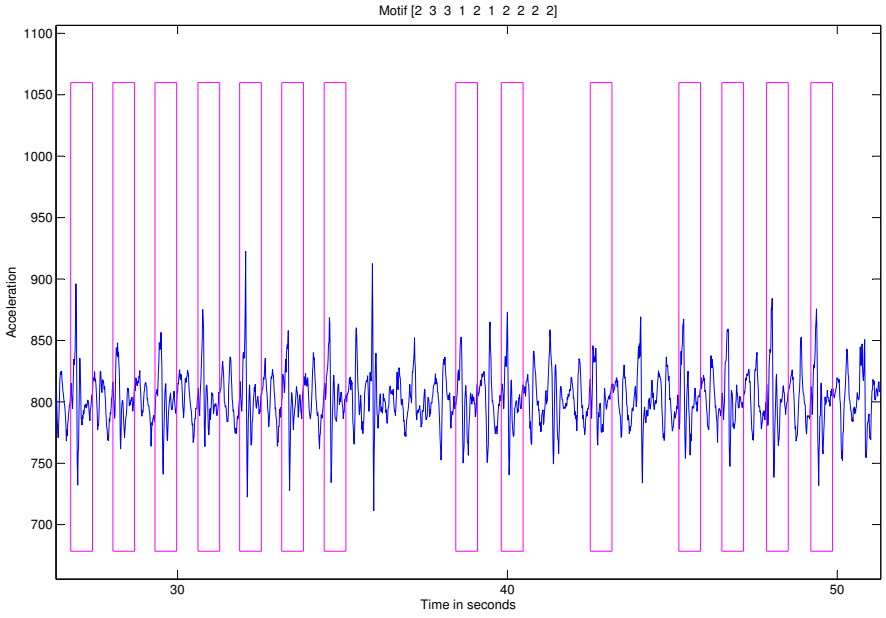

Figure 3.6: Zoom-in of best SAX motif using parameters $m = 50, w = 10, \alpha = 3$.

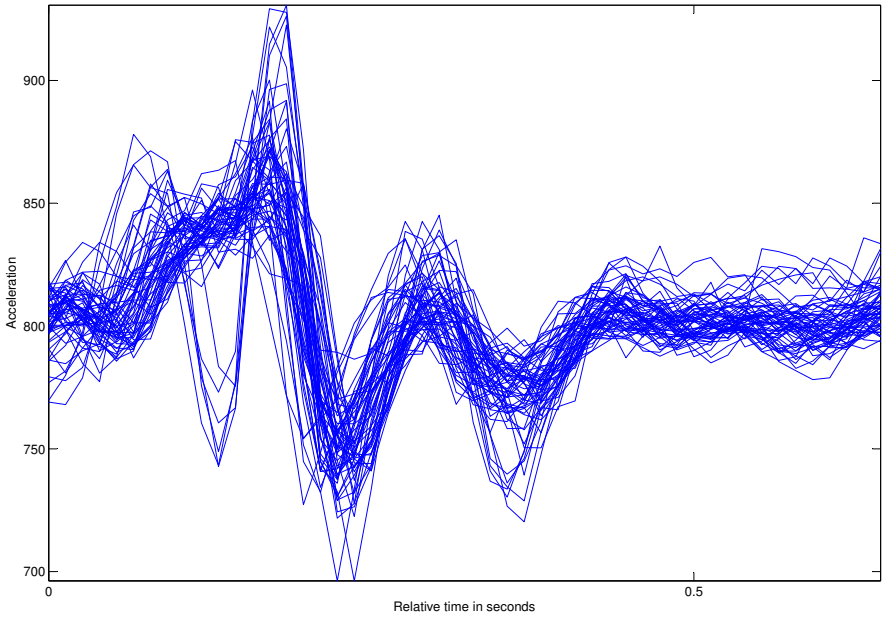

Figure 3.7: Plot of non-trivial subsequences matched as the best SAX motif using parameters $m = 50, w = 10, \alpha = 3$.

Due to using this distance measure we need to redefine matches and non-trivial matches. We use definitions similar to Chiu et al. (2003).

Definition 4. Given a real number $r \geq 0$, two subsequences S_i and S_j are a *match* if and only if

$$d(S_i, S_j) \leq r.$$

Definition 5. Given two real numbers $r \geq 0, c \geq 1$ and three subsequences S_i, S_j and S_k such that $j < k$, $d(S_i, S_j) \leq r$, and $d(S_i, S_k) \leq r$,
S_j and S_k are *non-trivial* matches of S_i if and only if

$$|j - k| \geq m \text{ and } \exists l \in \mathbb{N} : (j < l < k) \;\&\; (d(S_i, S_l) > c \cdot r)$$

Both distance and matching are symmetrical relations for any pair of subsequences. We illustrate non-trivial matching is not symmetrical. Given S_j and S_k are non-trivial matches of S_i does not imply S_j and S_k are a match. If they are not a match, then S_j cannot occur in a non-trivial matching for S_k and vice versa. Hence, for the ED measure, non-trivial matching can only be defined for triples of subsequences. Note it is possible to choose $i = j$ or $i = k$, in which case the relation *is* symmetrical. Note also the largest set containing non-trivial matches of S_i does not necessarily include S_i and may not be unique. To find a largest set of non-trivial matches we can use Algorithm 3.1 just as with SAX.

We select $c = \sqrt{2}$ implying the last test in the definition gives us the convenient test $d(S_i, S_k)^2 \geq 2 \cdot r^2$. Actually we never compute $d(S_i, S_j)$ but only

$d(S_i, S_j)^2$ because it does not involves taking a square root, which is a costly operation, and the test gives the same result.

The algorithm we use for finding motifs using ED is as simple as with finding motifs using SAX: first we cut the input data into pieces using a sliding window and then for each subsequence we count how many non-trivial matches it has. We run two experiments. In the first experiment we count non-trivial matches using the PAA representation and in the second experiment we count non-trivial matches using the representation after applying only the sliding window and normalization.

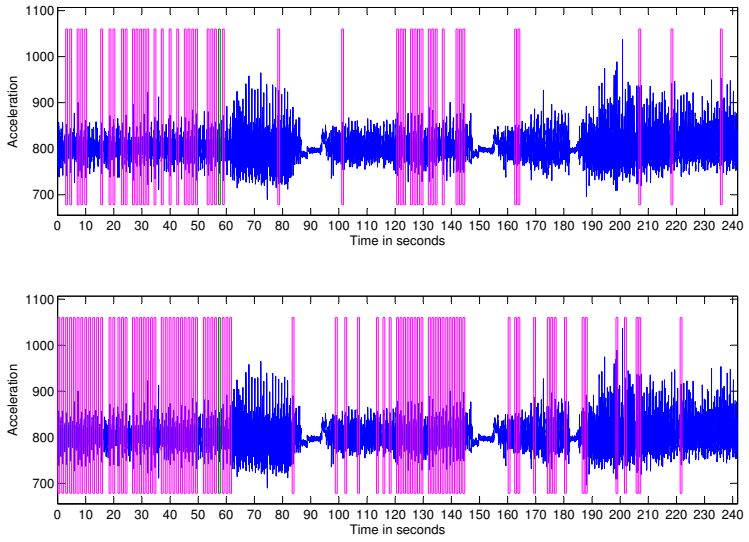

Figure 3.8: SAX (top) and ED (bottom) motif based on subsequence S_{4347}. The original subsequence is marked in green.

The parameter r determines the maximal distance between matching subsequences. In the first experiment we choose $r = 1.0$. We compute, for each subsequence, all non-trivial matches using the distance between the corresponding PAA vectors. The two best motifs found have 89 matches each. These motifs are based on subsequences S_{4347} and S_{10678}. In Figures 3.8 and 3.9 we compare the SAX and the ED motifs based on these subsequences. Consider Figure 3.8. Because of the chosen parameter r the SAX motif occurs less frequent. The motif is often found and at similar places for both methods. We conclude the difference between the SAX motif and the ED motif is small. Consider Figure 3.9. The difference between the SAX motif and the ED motif in this figure is big. The difference is even more explicit in Figure 3.10. In this Figure we see the preprocessed values corresponding to the non-trivial matches in Figure 3.9. ED gives a few matches for which it is questionable whether they actually are matches, but SAX clearly misses a lot of good matches. This is caused by the strict boundaries used in SAX. Any

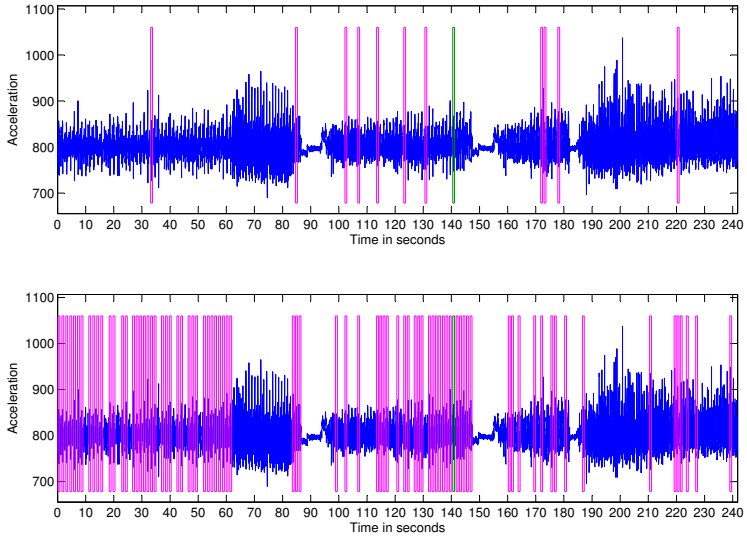

Figure 3.9: SAX (top) and ED (bottom) motif based on subsequence S_{10678}. The original subsequence is marked in green.

two observations that have only minor difference can be mapped to different characters and hence do not match the motif using the current algorithm.

For the second experiment we select $r = 4.0$ and compute for each subsequence the largest set of non-trivial matches. We use only the sliding window to compute subsequences and normalize each subsequence S_i to have $\mu(S_i) = 0$ and $\sigma(S_i) = 1$. The best motif is based on subsequence S_{9348}, which has 84 non-trivial matches. Figure 3.11 shows an overview of the non-trivial matches in the series. Figure 3.12 shows the corresponding subsequences. It occurs frequently in the series and gives a clear pattern, i.e. most matches are good matches.

3.4 Generalizations

It is interesting to know how well the motifs found in the experiments in Sections 3.2 and 3.3 generalize over people and over activities. Given in this thesis we build a generally applicable activity classifier, a good motif would generalize well over people but not over activities.

Let us consider generalization over people first. We concatenate the data annotated with *walking* from subjects one to five. We repeat the motif discovery experiment using SAX as in Section 3.2. The data set for the five subjects together is eight times as large as for subject one alone. In the experiment in Section 3.2 the best SAX motif occurred 59 times, so extrapolation would lead to $59 \cdot 8 = 472$ occurrences. The best motif found in the extended data

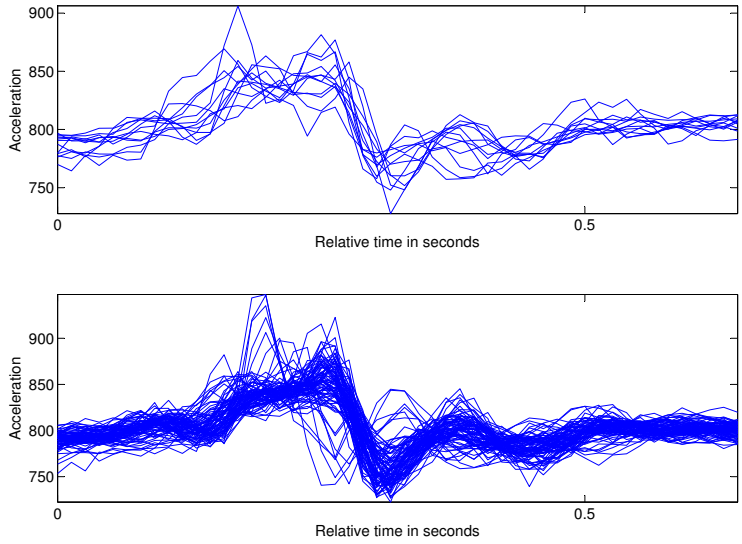

Figure 3.10: SAX (top) and ED (bottom) subsequences corresponding to motif based on subsequence S_{10678}.

set occurs 445 times, only slightly less than we would expect assuming perfect generalization. The result can be seen in Figure 3.13. We also find in the figure the motif occurs in all five parts of the data set. We conclude the motifs generalize well over people.

To see how well motifs generalize over different activities we concatenate data of subject 1 from the activities *walking*, *sitting*, *running*, *bicycling* and *vacuuming*. In Figure 3.14 we find the best motif in the data set, which has 640 non-trivial occurrences. It is the flat motif corresponding to no acceleration. Most of the data from *sitting* contains no acceleration and matches this motif. We also see the motif matches small parts of the other activities. This is not strange because in Section 2.5 we already showed various activities partly consist of no acceleration. For this chapter the data was only preprocessed, not cleaned. Since this flat motif is not very interesting we look for the best motif from *running*. This motif, shown in Figure 3.15, has 160 non-trivial matches. We see a few matches are in different activities, but mostly it matches subsequences from *running*. We conclude the motifs mostly match a single activity, which is precisely what we want.

3.5 Discords

For building a classifier we are interested in finding regularity in the data. For other purposes, such as fall-detection, we are also interested in finding deviations from the regular patterns. The opposite of a motif is called a *dis-*

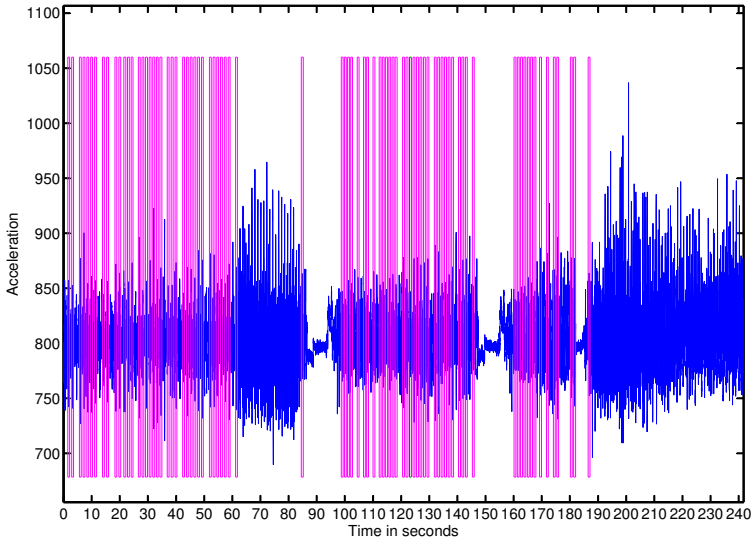

Figure 3.11: ED (without PAA) motif based on subsequence S_{9348}.

cord, which is a pattern deviating from all others. Hence, the search for these patterns is discord discovery. We now conduct some small experiments to find out whether the methods for finding motifs can also be used for finding discords.

For each of the experiments in Sections 3.2 and 3.3 we conduct a simple test. We count how many patterns have only a single non-trivial match. Under the definition of matching for both SAX and ED, any subsequence will match itself. Thus there are no subsequences with zero matches and a single non-trivial match is the lower bound. We summarize the results in Table 3.2.

Test	Average #matches	# Discords
$SAX, \alpha = 3$	16.23	3402
$ED_{PAA}, r = 1.0$	17.65	615
$ED_{noPAA}, r = 4.0$	22.43	1693
$SAX_{mindist_0}, \alpha = 8$	17.58	529

Table 3.2: Number of discords for each experiment.

In SAX a lot of subsequences are unique because of the inflexible breakpoints. In Lin et al. (2003) it is proven SAX strings can be used to give a lower bound on the Euclidean distance between PAA vectors. Two SAX strings $\widehat{S}_i$ and $\widehat{S}_j$ have a lower bound distance of 0 if and only if $|\widehat{s}_{i,k} - \widehat{s}_{j,k}| \leq 1 \ \forall k = 1, \ldots, m$, or in words, the k-th value of $\overline{S}_i$ is in the same as or an adjacent interval of the k-th value of $\overline{S}_j$ for all $k = 1, \ldots, m$. This is probably a more suitable

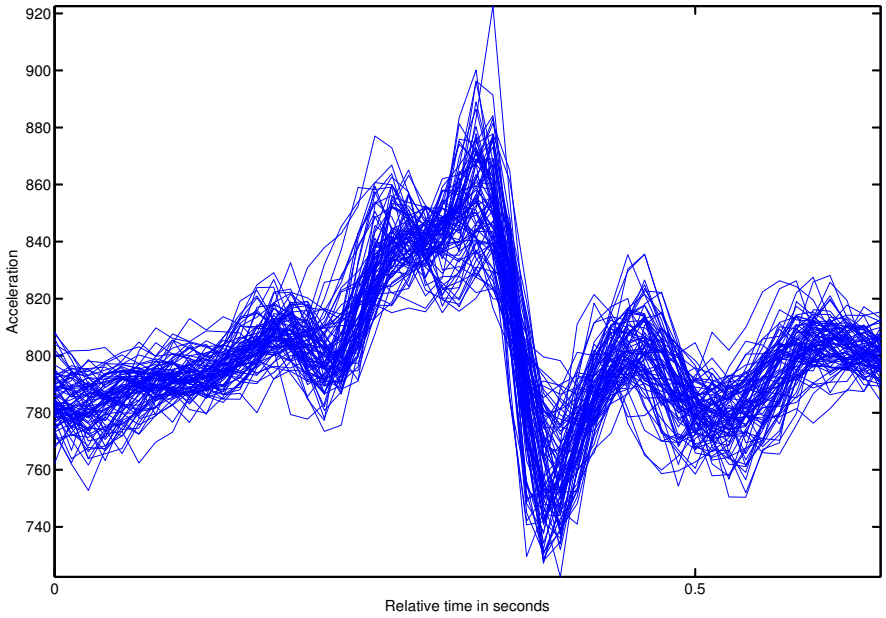

Figure 3.12: Subsequences corresponding to ED (without PAA) motif based on subsequence S_{9348}.

approach to find discords and the result is in the table as $SAX_{mindist_0}$. We find in the table the number of discords is indeed much lower while having approximately the same number of matches.

The number of discords is for each of the experiments very high. There is no measure available to assess how many anomalous patterns are actually in the data set, but there is also no reason to assume there are *any* discords in the data set. Clearly the parameters used in the experiments are set too strict for the purpose of detecting discords. We leave the search for parameters for anomaly detection open for further research.

3.6 Conclusion

Considering the motifs discovered in Sections 3.2 and 3.3, finding regular patterns in the given time series works well. In Section 3.4 we showed both numeric (ED) and discrete (SAX) motifs generalize well over people, but not over activities. We conclude both representations are useful as input to a classification algorithm to discriminate between activities. We use these results in Chapter 4 to build a classification algorithm.

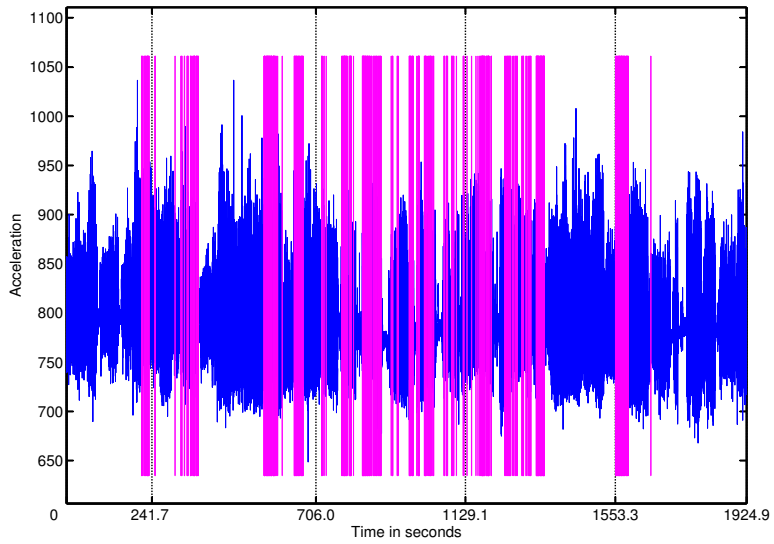

Figure 3.13: Occurrences of SAX motif $\{b, b, c, c, b, c, b, a, a, a\}$ in *walking* data from subjects 1–5.

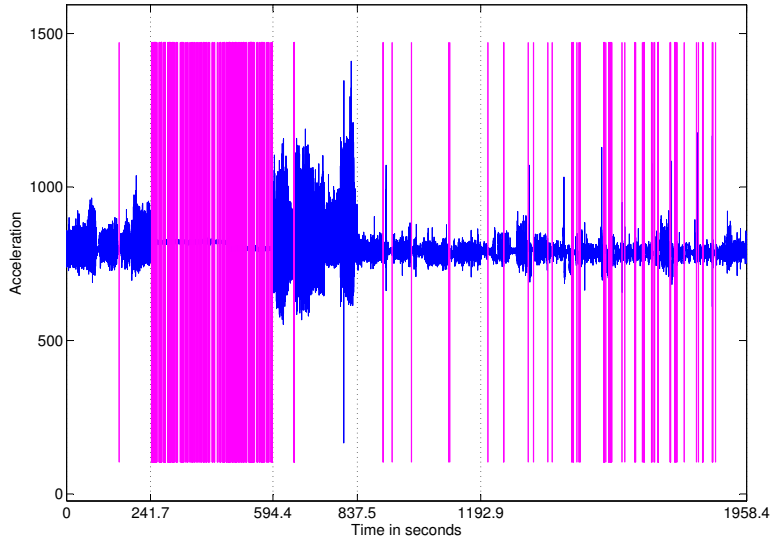

Figure 3.14: Occurrences of SAX motif $\{b, b, b, b, b, b, b, b, b, b\}$ in data from various activities.

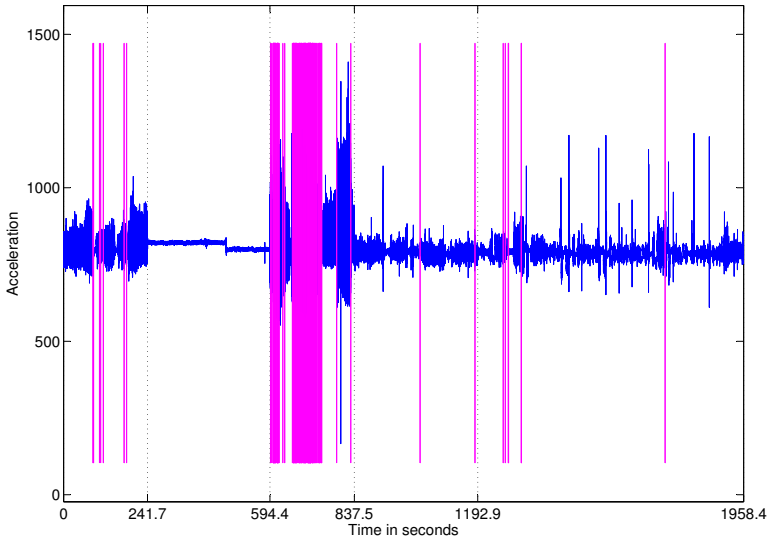

Figure 3.15: Occurrences of SAX motif $\{c, b, a, a, b, c, c, b, a, a\}$ in data from various activities.

Chapter 4

Compression-based Classification

Compared to the previous work presented in Chapter 1, we take a fairly different approach to building a classifier. In the data mining area of association analysis a lot of research has been done on finding frequent patterns in discrete data sets. Recent work shows great promise in using frequent patterns for describing a data set i.e. for classification or clustering (Van Leeuwen et al. 2006). We discuss this technique throughout Chapter 4 and show it also works well for classification of activities in Chapter 5. We start by adding some preprocessing steps.

4.1 Preprocessing revisited

Recall that after preprocessing (Section 2.3), our input data is given by

$$T = t_1, \ldots, t_n.$$

To use our classification algorithm we have to convert this time series to a discrete representation. The algorithm to convert the data consists of three stages inspired by SAX and the motif discovery results of Chapter 3.

1. **Normalization:** For each value subtract the moving average of the last 38 samples.
2. **Smoothing:** For each value t_i set $t_i \leftarrow \frac{t_{i-8}+\ldots+t_i+\ldots+t_{i+8}}{15}$.
3. **Discretization:** Convert each value to an appropriate character using a set of breakpoints.

We now explain each of these steps separately.

4.1.1 Normalization

The sliding window step used in Chapter 3 causes a lot of data multiplication. Taking into account all windowed data severely slows the process of building

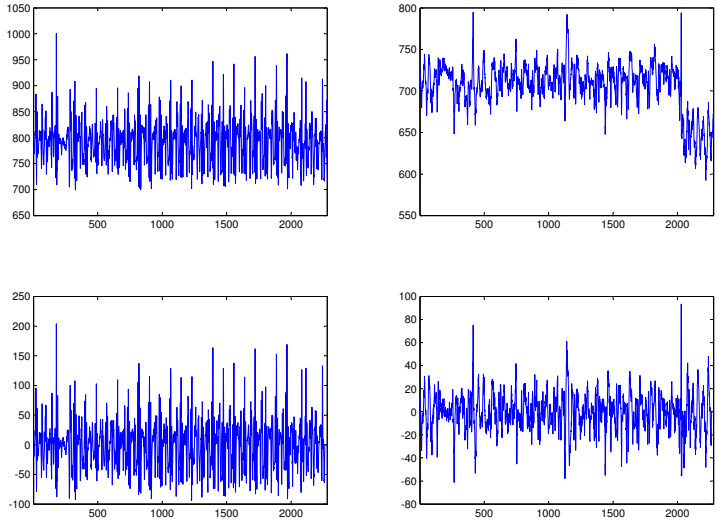

Figure 4.1: Top two graphs are samples of *walking* and bottom two graphs show the samples after normalization.

a classifier. A second 'problem' is normalizing the variance eliminates information about the intensity of the acceleration. This is not noise we want to remove, but a valuable resource for identifying the current activity. Therefore we choose to normalize each observation individually such that the mean of the entire series is zero, i.e. $\mu(T) = 0$, and leave the variance of the data as it is. In Figure 4.1 we find an example of normalizing two samples of *walking* data. We find in the figure the offset shift is removed while the general pattern of the data is unchanged.

4.1.2 Smoothing

Inspired by the intermediate PAA representation used in SAX we smooth each observation by taking the mean over fifteen adjacent values. The difference between this method and PAA is there is no dimensionality reduction here. Although it is probably a good idea to use some form of reduction we leave this open for further research. We illustrate the effect of smoothing in Figure 4.2.

4.1.3 Discretization

The discretization is the same as used in SAX. We compute beforehand a set of breakpoints $B = \{\beta_0, \ldots, \beta_\alpha\}$ and assign to each observation the character

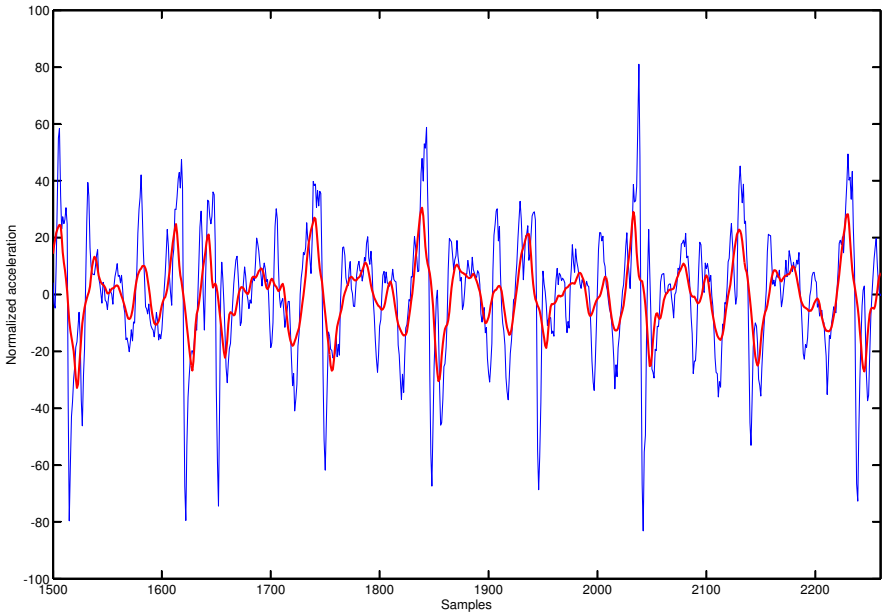

Figure 4.2: Effect of smoothing on a normalized sample of *walking* data.

associated with the interval. So

$$\widehat{t}_i \leftarrow a_j \quad \text{iff } \bar{t}_i \in [\beta_{j-1}, \beta_j]$$

In SAX the intervals all cover a region with equal probability. Given our data set it is possible to estimate such intervals but it would not reflect reality. Let us write C as the set of activities and let us rewrite the probability of some preprocessed observation $\bar{t}_i$ having a value in some interval $[\beta_{j-1}, \beta_j]$ as the sum over the marginal probabilities for the activities:

$$P\left(\bar{t}_i \in [\beta_{j-1}, \beta_j]\right) = \sum_{c \in C} P\left(\bar{t}_i \in [\beta_{j-1}, \beta_j] | \bar{t}_i = f(c)\right) \cdot P(c) \tag{4.1}$$

This shows a serious problem for estimating the breakpoints. The length of the samples for each activity does not reflect the a priori distribution of activities. Because we have no knowledge on the a priori distribution of activities in reality, it is impossible to compute equiprobable intervals.

We know we want our classification algorithm to discriminate between activities. Therefore the breakpoints have to be chosen such that the resulting representation still allows this. It is unknown how to pick good breakpoints and it will probably influence the quality of the classifier to a large extent. Therefore we will vary the breakpoints during the experiments. So for actual value choices we refer to Chapter 5.

4.1.4 Preprocessing summary

After adding these three steps the entire preprocessing phase contains four steps:

1. Compute the L^2-norm
2. Normalize using moving average
3. Smooth by averaging
4. Discretize using breakpoints

4.2 Krimp

In recent years several algorithms using frequent item sets for classification have been developed. One of those is *Krimp* (Van Leeuwen et al. 2006), which is based on the Minimum Description Length (MDL) principle. The idea behind MDL is best illustrated by a small quote from Grünwald (2005):

> *"The goal of statistical inference may be cast as trying to find regularity in the data. "Regularity" may be identified with "ability to compress." MDL combines these two insights by* viewing learning as data compression: *it tells us that, for a given set of hypotheses $\mathcal{H}$ and data set D, we should try to find the hypothesis or combination of hypotheses in $\mathcal{H}$ that compresses D most."*

MDL is an implementation of the Kolmogorov Complexity, which is defined as the shortest program to output D and then halts. Unfortunately, as shown by Li & Vitányi (1997), the shortest program is both uncomputable and suffers from bias because of specific constructs in languages etc. MDL tries to overcome these problems by restricting the possible models and choosing models that are generally applicable. A central idea in MDL is using two-part codes: $L(\mathcal{H})$, describing the cost of the hypothesis, and $L(D|\mathcal{H})$, describing the cost of the data given the hypothesis. We could say *Krimp* is a version of MDL for specific inputs: databases containing discrete values. The time series we now have after preprocessing fit perfectly to this algorithm. We now discuss how the algorithm works.

4.2.1 Generating Frequent Patterns

The first step to capturing the characteristics of a database is generating all frequent patterns. In the next step (Section 4.2.2) we make a selection of the frequent patterns and link them to codes for the compression. We need some definitions to understand precisely what is happening.

Definition 6. A *pattern* is a series of characters

$$p = \{p_1, \ldots, p_m\} \quad \forall p_i \in p : p_i \in A$$

Denote $|p| = m$ the *length* of the pattern.

Definition 7. A pattern $p = \{p_1, \ldots, p_m\}$ *occurs* in series $T = t_1, \ldots, t_n$ at position i if and only if

$$\forall j \in \{1, \ldots, m\} : p_j = t_{i+j-1}$$

Definition 8. Given a pattern $p = \{p_1, \ldots, p_m\}$ and a series $T = t_1, \ldots, t_n$, the set $X(p)$ is the *largest non-trivial set* of indices where p occurs in T. The set $X(p)$ is *non-trivial* if and only if

$$\forall i, j \in X(p) : |i - j| \geq m$$

Denote the cardinality of $X(p)$ as $|X(p)|$, which equals the number of unique indices in the set. We also refer to this as the *support*. Note the largest non-trivial set can, like in Section 3.3, be computed from the set of all indices where p occurs in T using a greedy algorithm. The property of non-triviality is used to make sure we do not over-count the number of occurrences of a pattern in the series. We illustrate this with a small example. Given a series $T = \{a, a, a, a, a\}$ the pattern $p = \{a, a\}$ occurs at indices 1,2,3 and 4. During encoding we want each observation to be encoded only once. Given that restriction, the pattern fits only twice in the series. Our definition gives the same answer ($|X(p)| = 2$). We are now ready to define the most important property.

Definition 9. Given a positive integer $minsup$, a pattern p is *frequent* in series T if and only if $|X(p)| \geq minsup$.

As said we want an algorithm to generate all frequent patterns. Now that we know exactly what that means we can build such an algorithm. To build an algorithm we use two important properties deduced from the series T being ordered:

1. If pattern $p = \{p_1, \ldots, p_m\}$ occurs in T at position i then one of the extensions $\{p_1, \ldots, p_m, p_{m+1}\}$ also occurs at position i, unless the occurrence is at the end of T.

2. And vice versa, if pattern $p = \{p_1, \ldots, p_m, p_{m+1}\}$ occurs in T at position i the shorter version $\{p_1, \ldots, p_m\}$ occurs in T at position i as well.

In the algorithm we use two tables with the same structure. One (denoted FP) contains the frequent patterns and the other (denoted CP) contains the candidates for expansion. The tables contain the patterns on the left hand side and *all* the indices where the pattern occurs in T at the right hand side.

p_1	$I(p_1)$
p_2	$I(p_2)$
$\ldots$	$\ldots$

As input FP and CP contain the singleton patterns $p_1 = \{a_1\}$, $p_2 = \{a_2\}, \ldots, p_\alpha = \{a_\alpha\}$ and their associated index sets $I(p_1), I(p_2), \ldots, I(p_\alpha)$. To generate all frequent patterns we use Algorithm 4.1, where the function *nontrivialSize* in line 8 returns the size of the set of non-trivial matches returned by the algorithm *nonTrivial* in Section 3.2.

The algorithm returns precisely all frequent patterns, because of the discussed properties deduced from T being an ordered series. It is also very

Algorithm 4.1 Generating the set of frequent patterns in time series T.

AllFrequentPatterns(T, FP, CP)
1 **while** $CP \neq \emptyset$
2 **do** $(p, I(p)) \leftarrow (p, I(p)) \in CP$
3 $CP.remove\,(p, I(p))$
4 $m \leftarrow length(p)$
5 $E \leftarrow T\,(I(p) + m)$ $\#I(p) + m$ is a list of indices
6 **for** $i \leftarrow 1$ **to** α
7 **do** $I(\{p_1, \ldots, p_m, i\}) \leftarrow \{e \in E : e = i\}$
8 **if** $nontrivialSize\,(I(\{p_1, \ldots, p_m, i\})) \geq minsup$
9 **then** $CP.add(\{p_1, \ldots, p_n, i\}, I(\{p_1, \ldots, p_n, i\})$
10 $FP.add(\{p_1, \ldots, p_n, i\}, I(\{p_1, \ldots, p_n, i\})$
11 **return** FP

efficient: each candidate is generated only once (based on property 1) and only all relevant candidates are generated (based on property 2). Note in this algorithm the tables FP and CP are not ordered in any specific way. Therefore when adding patterns they can be added in the most computationally efficient way (i.e. at the end of the table). Note that any pattern added to CP is frequent. So using just one ordered table is also possible, but this way it is easier to adapt the algorithm to generating restricted sets of frequent patterns, which we discuss next.

Given a low value for $minsup$ there is a *vast* set of frequent items. Although it is the job of *Krimp* to pick out a good set of patterns and it might be unrivaled in terms of quality of the result, it is also very costly in terms of computation time. We will come back to the computation time in Chapter 5. There exist some simple approaches to reduce the size of the frequent pattern set, and we use one of them in the experiments and compare the results to the results of using the full set in Chapter 5.

Definition 10. Given a positive integer $minsup$, a pattern $p = \{p_1, \ldots, p_n\}$ is *maximally frequent* in series T if and only if

$$|X(p)| \geq minsup \text{ and } \neg\exists q = \{p_1, \ldots, p_n, q_{n+1}\} : |X(q)| \geq minsup$$

In simple words p is maximally frequent if it is frequent and there is no character we can extend p by such that the extended pattern q is frequent. We generate the set of maximally frequent items by using the algorithm for all frequent patterns and only adding a pattern to FP if no pattern is added to CP in the for-loop, instead of adding each candidate pattern directly to FP.

Having defined and generated the set of frequent patterns we move on to the learning phase of *Krimp*.

4.2.2 Building Code Tables

As with many classification algorithms the characteristics of the classes have to be extracted first. In *Krimp* this means building a code table for each class. The classification itself can be done very fast using the precomputed code tables. A code table consists of two columns:

Pattern	Code-length
$\{a\}$	$L(\{a\})$
$\{b\}$	$L(\{b\})$
$\{c\}$	$L(\{c\})$
$\{a,b\}$	$L(\{a,b\})$
$\{c,a\}$	$L(\{c,a\})$
...	...

The code table (CT) always contains all possible singleton patterns (i.e. the unique characters) at the top. The other patterns in the table are ordered ascending first by length and secondly by support. Before we can build a code table we must first discuss the coding scheme.

Compression of the time series is achieved by converting the input series to a list of codes. *Krimp* uses a lossless compression scheme, meaning there exists a function mapping the codes back to the original series. For such a function to exists the coding must have the following three properties.

1. The codes must cover the entire input series; each of the observations must be encoded.
2. The codes cannot overlap each other; no observation is allowed to be encoded by multiple codes.
3. Each code must be unique and correspond to a single pattern.

Any coding scheme having these three properties produces a unique and invertible mapping from input data to codes and is a lossless compression scheme.

The coding algorithm used in *Krimp* works as follows. To encode T we take the last pattern in CT that occurs in T, replace the covered part by the code and recursively encode the remaining parts of T until it is entirely encoded. An example: given $T = \{t_1, \ldots, t_{i-1}, t_i, \ldots, t_j, t_{j+1}, \ldots, t_n\}$ and a pattern p occurring at i (so $p = \{t_i, \ldots, t_j\}$), encode $t_i, \ldots, t_j$ with the code associated with p and recursively encode $\{t_1, \ldots, t_{i-1}\}$ and $\{t_{j+1}, \ldots, t_n\}$.

Actually we are not interested in the actual codes, but only in the lengths of the codes. The code lengths should be chosen such that the length of the entire code sequence is minimized. From information theory (Grünwald 2005) we know the optimal length of a code is given by

$$\ell_{CT}(p_i) = -log\left(P(p_i|T)\right) = -log\left(\frac{freq(p_i)}{\sum_{p_j \in CT} freq(p_j)}\right). \tag{4.2}$$

Where $freq(p_i)$ (the *frequency* of pattern p_i) is the number of times it is used in the encoding. This is not the same as the support, which equals the number of times p_i occurs in T. We can also compute the length of the entire sequence of codes by

$$L_{CT}(T) = -\sum_{p_i \in CT} freq(p_i) \cdot log\left(\frac{freq(p_i)}{\sum_{p_j \in CT} freq(p_j)}\right). \tag{4.3}$$

MDL theory tells us we have to penalize any hypothesis for the complexity of the model, in this case the size of the code table. Clearly the cost for storing

the codes in the right column of the code table should equal the length of the codes. For the left column we use the length of each pattern encoded with the *standard table (ST)*, which is the initial code table containing only the singleton patterns. The size of the entire table now becomes

$$L(CT) = \sum_{p_i \in CT} \ell_{ST}(p_i) + \ell_{CT}(p_i). \tag{4.4}$$

It has been proven by Siebes et al. (2006) that the optimal set of patterns is the set minimizing

$$L(CT, T) = L(CT) + L_{CT}(T). \tag{4.5}$$

Any subset (of the set of frequent patterns) which contains the singleton patterns is a valid code table. Since the frequent pattern set is very large, it is infeasible to compute $L(CT, T)$ for each possible code table. Therefore we use a simple heuristic algorithm called Naïve-Compression (Van Leeuwen et al. 2006) to find a good code table. We order the set of frequent patterns descending first by support and secondly by length. Now we try for each pattern – in order – to insert it into the code table at the right place and recompute the total size $L(CT, T)$. If the new size $L(CT, T)$ is smaller, the pattern stays in the table, otherwise we remove it again.

4.3 Classification Algorithm

The classification algorithm is based on the principle that a series belonging to a certain class $c_i \in C$ will be compressed well by the code table associated with that class. Using the Naïve Bayes assumption we expect the following equation to be true (Van Leeuwen et al. 2006)

$$\text{If } \ell_{CT_i}(T) < \ell_{CT_j}(T) \text{ then } P(T|c_i) > P(T|c_j) \tag{4.6}$$

Here we define the coded length as the sum of the codes used in the encoding

$$\ell_{CT_i}(T) = \sum_{p_j \in CT_i} -log\left(\frac{freq(p_j)}{\sum_{p_k \in CT_i} freq(p_k)}\right) \tag{4.7}$$

This leads to the classification Algorithm 4.2.

Algorithm 4.2 Classifying series T using the list of Code Tables CT.

```
CLASSIFY(T, CT)
1  class ← 0
2  l_min ← ∞
3  for i ← 1 to #classes
4  do l_i ← ℓ_{CT_i}(T)
5     if l_i < l_min
6        then class ← i
7             l_min ← l_i
8  return class
```

Chapter 5

Experimental Results

In this section we review the results of *Krimp* as an activity classifier. In Section 5.1 we discuss setting the parameters for the experiments. In Section 5.2 we show the result for the entire data set. In Section 5.3 we look at a subset of activities where *Krimp* achieves higher accuracy than other algorithms. We discuss some slightly altered experimental setups in Section 5.4. In Section 5.5 we review the results of detecting irregularities with *Krimp*. Finally, we improve the classification results from Section 5.5 by introducing flexible matching in Section 5.6.

5.1 Test Set-Up

Table 5.1 summarizes the parameters used in the experiments. We motivate each choice separately.

Parameter	Setting
Cross-validation	Leave one subject out
Window length	760 samples (= 10 seconds)
Type Frequent Item	All vs. Maximal Only
Minimum Support	100% to 0.05%
Breakpoints	$[-\infty, -34, -30, \ldots, -2, 2, \ldots, 30, 34, \infty]$,
	$[-\infty, -33, -27, \ldots, -3, 3, \ldots, 27, 33, \infty]$,
	$[-\infty, -36, -28, \ldots, -4, 4, \ldots, 28, 36, \infty]$,
	$[-\infty, -35, -25, -15, -5, 5, 15, 25, 35, \infty]$,
	$[-\infty, -30, -18, -6, 6, 18, 30, \infty]$,
	$[-\infty, -35, -21, -7, 7, 21, 35, \infty]$

Table 5.1: Settings used throughout this chapter.

We want the classifier to work accurately for unknown subjects. Leave one subject out cross-validation gives the most accurate result regarding the real performance of the classifier.

To detect precisely when an activity starts and ends the window length should be as small as possible. In Bao & Intille (2004) a window of 512 samples

($\approx$ 6.7 seconds) is used to ensure each window contains some repetition of the basic movements involved in the activity. We make a safe choice of 760 samples (= 10 seconds) with no overlap between windows.

As explained in Section 4.2.1, reducing the set of all frequent patterns speeds up the process of building the classifier, but may reduce the accuracy of the classification algorithm. We consider the full frequent pattern set and the set of maximal frequent patterns. When the set is not explicitly mentioned we use the full frequent pattern set.

Cycles of movement, for example two steps while walking, can take up to 1.5 seconds (= 100 samples). So even a pattern covering the entire data set perfectly has a frequency of 1% in the data set. We expect a minimum support of 1% or lower is required for a good classifier. We vary the minimum support from 100% (equals use *Standard Code Table*) to 0.025%. To be able to compare unequally sized data sets fairly we use relative minimum support.

It is unknown what the effect on the accuracy of the classifier is of choosing certain breakpoints, therefore we simply use a number of fixed sized intervals. Only data annotated with *running* exceeds -30 and $+30$ making the chosen interval ranges broad enough for recognition of any activity.

5.2 Results All Activities

We use the regular definition of *accuracy* measured over the entire data set.

$$accuracy = \frac{\text{\#observations correctly classified}}{\text{\#observations}} \tag{5.1}$$

And the regular definitions of *precision* and *recall* measured for each class separately.

$$precision = \frac{\text{\#observations of class } i \text{ classified as } i}{\text{\#observations classified as } i} \tag{5.2}$$

$$recall = \frac{\text{\#observations of class } i \text{ classified as } i}{\text{\#observations of class } i} \tag{5.3}$$

To compare our experimental results we give the full classification matrix from Bao & Intille (2004) in Table 5.2. The total accuracy is 50.1%. Precision and recall is not very high for most activities and for some, i.e. *climbing stairs* and *strength-training*, both precision and recall is very low. We assume there are two causes for not too good results. There are annotation errors in the data set, meaning large parts of *riding elevator*, *riding escalator* is actually data where the subject is walking or standing. Similar problems occur for the zero-acceleration classes (*sitting*, *working on PC* etc.). When looking at the input data there are clearly some parts not annotated correctly. In some cases the classification algorithm may have produced the true answer but in the scoring is it counted as incorrect. The second cause for disappointing results is that the classification algorithm can be improved. The goal of this chapter is to prove that.

As explained in Chapter 2 the assumption of unknown orientation was not used in the paper of Bao & Intille (2004). We expect that some activities like *sitting*, *eating* and *working on PC* are impossible to distinguish from each other

Class	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	Recall %
(a) Walking	615	127	0	0	15	10	0	0	40	21	7	25	11	51	3	0	22	47	43	13	58.6
(b) Carrying	160	271	6	2	66	16	2	2	42	50	31	109	23	129	50	1	27	195	223	36	18.8
(c) Sitting	2	0	480	75	1	9	59	49	0	0	5	20	9	0	0	93	0	0	2	0	59.7
(d) Working on PC	0	0	198	559	3	29	60	42	1	3	10	2	1	1	2	2	0	0	1	2	61.0
(e) Standing	3	5	1	2	659	14	0	2	1	38	16	8	13	9	9	2	9	4	4	10	81.5
(f) Eating/drinking	6	1	18	34	19	562	15	5	0	7	12	2	41	8	16	5	43	2	2	5	70.0
(g) Watching TV	0	2	44	21	1	6	229	82	0	1	1	0	0	0	1	25	1	0	0	2	55.0
(h) Reading	1	2	140	71	30	28	180	343	13	0	14	24	5	22	8	159	2	3	0	2	32.8
(i) Running	63	48	0	0	5	3	0	0	375	12	7	5	4	9	3	0	10	6	5	5	67.0
(j) Bicycling	8	20	1	1	33	9	1	0	1	668	16	10	10	31	7	0	16	3	3	24	77.5
(k) Stretching	14	15	16	21	33	26	25	24	2	29	291	34	28	69	42	3	36	7	18	13	39.0
(l) Strength-training	25	42	18	24	30	31	31	39	3	29	60	61	16	89	32	9	22	16	20	9	10.1
(m) Scrubbing	5	6	0	2	21	51	1	6	2	27	17	7	169	33	51	2	77	4	7	9	34.0
(n) Vacuuming	7	5	3	4	15	12	10	18	4	9	32	9	16	726	20	0	16	2	3	7	79.1
(o) Folding laundry	4	13	2	9	28	36	3	3	4	5	56	7	20	24	570	5	52	2	15	6	66.0
(p) Lying down	0	0	3	2	17	29	14	13	0	2	2	26	2	5	1	712	3	0	2	0	85.5
(q) Brushing teeth	11	9	0	1	64	86	1	1	4	22	28	13	100	66	57	0	224	5	19	36	30.0
(r) Climbing stairs	108	118	0	0	9	8	0	2	13	15	11	13	14	37	5	0	12	6	19	20	1.5
(s) Riding elevator	109	118	0	1	53	82	0	6	6	27	36	52	60	88	67	3	71	44	276	53	24.0
(t) Riding escalator	8	5	0	0	10	10	0	1	2	13	16	8	5	19	4	1	14	4	5	55	30.6
Precision %	53.5	33.6	51.6	67.4	59.3	53.2	36.3	53.8	73.1	68.3	43.6	14.0	30.9	51.3	60.1	69.7	34.1	1.7	41.4	17.9	

Table 5.2: Full classification matrix from Bao & Intille (2004)

under that assumption. The subject's posture can be recognized if the orientation of the two axis compared to the body is known. This is actually the only information available from the hip sensor in the aforementioned activities.

Class	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	Recall %
(a) Walking	115	37	0	0	0	0	0	0	0	2	1	3	0	0	0	0	0	2	1	0	71.4
(b) Carrying	41	114	0	0	0	0	0	0	0	2	0	11	0	2	0	0	0	5	0	0	65.1
(c) Sitting	0	0	51	59	15	10	2	3	0	0	0	0	0	0	6	18	15	0	0	0	28.5
(d) Working on PC	0	0	66	75	10	9	5	7	0	0	0	0	0	0	6	29	10	0	0	0	34.6
(e) Standing	0	0	32	29	19	9	4	4	0	6	0	0	0	22	5	21	7	0	0	0	12.0
(f) Eating/drinking	0	1	26	14	17	7	5	7	0	0	0	5	1	8	33	12	36	1	0	0	4.0
(g) Watching TV	0	0	40	50	11	3	2	4	0	0	0	0	0	6	8	23	3	0	1	1	1.3
(h) Reading	0	0	64	68	11	4	2	7	0	0	0	0	0	7	10	28	8	0	1	1	3.3
(i) Running	0	0	0	0	0	0	0	0	70	0	0	0	0	0	0	0	0	0	0	0	100.0
(j) Bicycling	2	0	0	0	0	0	0	0	0	88	0	7	0	22	0	0	1	0	0	0	73.3
(k) Stretching	5	5	0	0	0	0	0	0	0	0	0	4	1	7	0	0	0	1	0	0	0.0
(l) Strength-training	11	2	0	0	0	0	0	0	0	2	1	28	0	16	0	0	0	7	1	0	41.2
(m) Scrubbing	1	0	0	0	0	0	0	0	0	2	1	3	0	6	1	0	1	0	0	0	0.0
(n) Vacuuming	1	5	0	0	4	2	0	0	0	37	2	44	0	105	18	0	8	5	4	0	44.7
(o) Folding laundry	3	1	5	3	19	6	2	4	0	2	0	8	0	41	74	4	41	0	0	0	34.7
(p) Lying down	0	0	79	70	15	10	1	4	0	0	0	0	0	0	10	14	5	0	2	1	6.6
(q) Brushing teeth	1	0	10	7	15	13	3	1	0	11	0	3	1	28	41	7	42	0	1	1	22.7
(r) Climbing stairs	9	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33	0	0	57.9
(s) Riding elevator	14	10	2	1	4	5	0	0	0	1	0	9	0	5	6	2	11	0	13	0	15.7
(t) Riding escalator	2	4	5	6	8	9	1	0	0	0	0	1	0	11	6	7	8	0	5	2	2.7
Precision %	56.1	58.8	13.4	19.6	12.8	8.0	7.4	17.1	100.0	57.5	0.0	22.2	0.0	36.7	33.0	8.5	21.4	61.1	44.8	33.3	

Table 5.3: Full classification matrix using *Krimp*

Consider the results of our algorithm in Table 5.3. For the experiment a minimum support of 0.10% is used, breakpoints are set to $[-\infty, -36, -28, \ldots, -4, 4, \ldots, 28, 36, \infty]$ and only the data of the first five subjects is used, because of the duration of the test. Overall accuracy is 30.9%. For the activities *walking*, *carrying*, *running*, *strength-training* and *climbing stairs*, *Krimp* scores better on both precision and recall. For *running* our classifier achieves an amazing 100.0% precision and recall. For *riding elevator* and *riding escalator* the precision is better but the recall is worse. For *bicycling*, *brushing teeth*, *vacuuming* and *folding laundry* the precision and recall are slightly worse. For the other activities the results are much worse. We already expected this phenomenon for the activities where this is no acceleration at all, but it also occurs in some low acceleration activities.

Although we do not have the decision tree results under the unknown orientation assumption we review the results for boxing the activities into 4 categories ranging from passive to intense activities. We use the categories listed in Table 5.4.

Category	Activities
Very low intensity	Sitting, Working on PC, Standing, Eating/drinking, Watching TV, Reading, Folding Laundry, Lying down, Brushing teeth, Riding elevator and Riding escalator
Low intensity	Bicycling, Stretching, Strength-training, Scrubbing and Vacuuming
Medium intensity	Walking, Carrying and Climbing stairs
High intensity	Running

Table 5.4: Activities categorized based on acceleration intensity.

Class category	1	2	3	4	Recall %
(1) Very low intensity	7203	974	363	31	84.0
(2) Low intensity	952	2486	179	12	68.5
(3) Medium intensity	612	547	1647	95	56.8
(4) High intensity	0	0	0	375	100.0
Precision %	82.2	62.0	75.2	73.1	

Table 5.5: Boxed results from Bao & Intille (2004).

Class category	1	2	3	4	Recall %
(1) Very low intensity	1644	176	37	0	88.5
(2) Low intensity	40	376	45	0	81.6
(3) Medium intensity	1	21	371	0	94.4
(4) High intensity	0	0	0	70	100.0
Precision %	97.6	65.6	81.9	100.0	

Table 5.6: Boxed results from *Krimp*.

In Table 5.5 we find the results from Bao & Intille (2004) boxed into the four categories and in Table 5.6 we find the results from *Krimp* boxed into the four categories. The overall accuracy is 75.7% and 88.5% respectively. While the performance of *Krimp* was significantly worse in the full activity set, here the results from *Krimp* are superior in all categories. Important to note is both these boxed results show us very well that misclassifications are made mainly between activities in the same category. Note the number of observations is so much lower because we only use data from the first five subjects and a longer window with no overlap between windows.

We have seen for the activities in the very low intensity category our approach is not going to achieve satisfactory accuracy, while for other activities (medium/high categories) our first attempt is already superior. For some of the activities in the very low and low intensity categories it was already identified in Bao & Intille (2004) higher level analysis (for example temporal reasoning) is required to robustly recognize these activities. In the next section we focus on boosting the accuracy for the medium and high intensity classes.

5.3 Results Ambulatory Activities

Parameter	Value				
Breakpoints	$[-\infty, -36, -28, \ldots -4, 4, \ldots, 28, 36, \infty]$				
Minimum support	0.10%				
Class category	a	b	i	r	Recall %
(a) Walking	119	39	0	3	73.9
(b) Carrying	42	128	0	5	73.1
(i) Running	0	0	70	0	100.0
(r) Climbing stairs	9	15	0	33	57.9
Precision %	70.0	70.3	100.0	80.5	

Table 5.7: Results for ambulatory activity from *Krimp*.

We start by looking at the test results in Table 5.7. The results are gathered using the same parameters as in Section 5.2, but with a data set consisting only the four activities from the medium and high intensity categories. The overall accuracy is 75.6%, which is not bad considering the activities are very similar. We are interested in the effect of the parameters to determine how we can improve this result.

Let us first consider changing the minimum support value. From the set of all patterns *Krimp* chooses a set of patterns that describes (compresses) the data well. Although we use the heuristic Naïve-Compression (see Section 4.2.2) to build the Code Tables, we expect adding patterns to the frequent pattern set can only improve the result. This is even more clear when we take into account the patterns are ordered by support before inserting them into the Code Table. For example if a minimum support of 0.20% gives 200 frequent patterns and a minimum support of 0.10% gives 250 frequent patterns, the Code Table for both sets is the same after evaluating 200 patterns. The 50 extra patterns have a support between 0.20 and 0.10% and are inserted later than the 200 frequent patterns with support $\geq$ 0.20%. Assuming ability to compress equals ability to classify, we expect a lower support always yields a better result.

As explained in Section 5.1 we do not know whether the character intervals determined by the breakpoints should be smaller or bigger, so let us look at the results in Table 5.8. For four of the breakpoint settings (1, 4, 5 and 6) the highest accuracy is achieved using the lowest minimum support value (0.05%), and for the other two breakpoint settings the highest accuracy is achieved with the second lowest minimum support value (0.10%). We observe that in general a lower minimum support increases the accuracy. In contrast with what we expected, it does not increase monotonically.

Considering the differences between the breakpoints we see for high minimum support (1.60% to 0.20%) the smaller intervals (1 to 3) are superior, while for low minimum support (0.10% and 0.05%) the larger intervals (5 and 6) perform better. These results are not easy to explain, but in the remainder of this section we try to get some additional insight.

To make the results more comparable let us look at Figure 5.1, which shows a graphical overview of the frequent pattern sets' and Code Tables' sizes. We

Breakpoints	Minsup	Accuracy
(1) $\{-34, -30, \ldots, -2, 2, \ldots, 30, 34\}$	1.60%	71.1%
	0.80%	71.7%
	0.40%	70.8%
	0.20%	73.2%
	0.10%	74.1%
	0.05%	75.6%
(2) $\{-33, -27, \ldots, -3, 3, \ldots, 27, 33\}$	1.60%	67.0%
	0.80%	67.4%
	0.40%	71.5%
	0.20%	72.8%
	0.10%	75.4%
	0.05%	74.3%
(3) $\{-36, -28, \ldots, -4, 4, \ldots, 28, 36\}$	1.60%	73.4%
	0.80%	69.1%
	0.40%	72.6%
	0.20%	74.5%
	0.10%	75.6%
	0.05%	75.2%
(4) $\{-35, -25, \ldots, -5, 5, \ldots, 25, 35\}$	1.60%	71.3%
	0.80%	64.1%
	0.40%	70.4%
	0.20%	71.1%
	0.10%	72.8%
	0.05%	75.4%
(5) $\{-30, -18, -6, 6, 18, 30\}$	1.60%	61.3%
	0.80%	55.1%
	0.40%	64.4%
	0.20%	70.8%
	0.10%	76.9%
	0.05%	78.0%
(6) $\{-35, -21, -7, 7, 21, 35\}$	1.60%	67.6%
	0.80%	67.8%
	0.40%	69.5%
	0.20%	69.3%
	0.10%	74.1%
	0.05%	80.1%

Table 5.8: Overview of all classification results.

notice for each of the breakpoint settings the number of candidate patterns grows linearly as the support decreases (note both axis have logarithmic scale). Each time the minimum support is halved the number of candidates becomes ~ 2.3 times as big. At equal minimum support level the number of candidate patterns is higher for bigger breakpoint intervals. Between breakpoint settings 1 and 6 the difference is a factor two. For the Code Table sizes the inverse holds. Bigger breakpoint intervals give a smaller Code Table.

To explain this let us reason from the perspective of the number of charac-

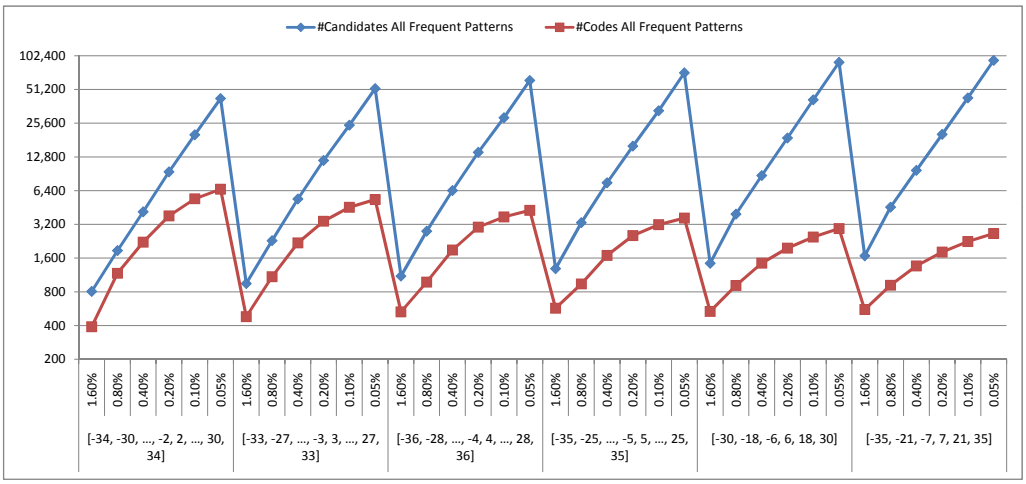

Figure 5.1: Size of full frequent pattern sets and Code Tables for each of the parameter settings.

ters. The number of characters for each of the breakpoint settings is 19, 13, 11, 9, 7 and 7 respectively. It is not strange there are longer patterns in the data set if we use a smaller alfabet to describe the input values, thus the number of candidate patterns grows. It is also not strange we need less patterns to give a good description of a database with a smaller alfabet. So the results of Figure 5.1 are easily explainable. This insight implies the use of a bigger alfabet requires a lower setting for the minimum support to have equally long patterns. Which in turn means comparison of breakpoints at a fixed minimum support level is not entirely fair. We verify this hypothesis by looking directly at the pattern lengths.

In Figure 5.2 we compare for two settings of breakpoints and two settings of minimum support the length of the patterns and their coverage of the data set. We see the coverage for *walking*, *carrying* and *climbing stairs* is very similar in all four cases. The area above (and left of) each curve tells us how long the patterns in the final Code Table are, weighed by the amount of data they encode. We see this area is bigger in the bottom two graphs, compared to the top two and also a lower minimum support gives a larger area above the curve. Looking closely we also see the area above the curve in the top right graph is smaller than in the bottom-left graph, for each of the four activities. This confirms our assumption a smaller alfabet gives much longer patterns in the Code Table and makes the comparison at fixed minimum support unfair. We keep this in the back of our mind while interpreting results.

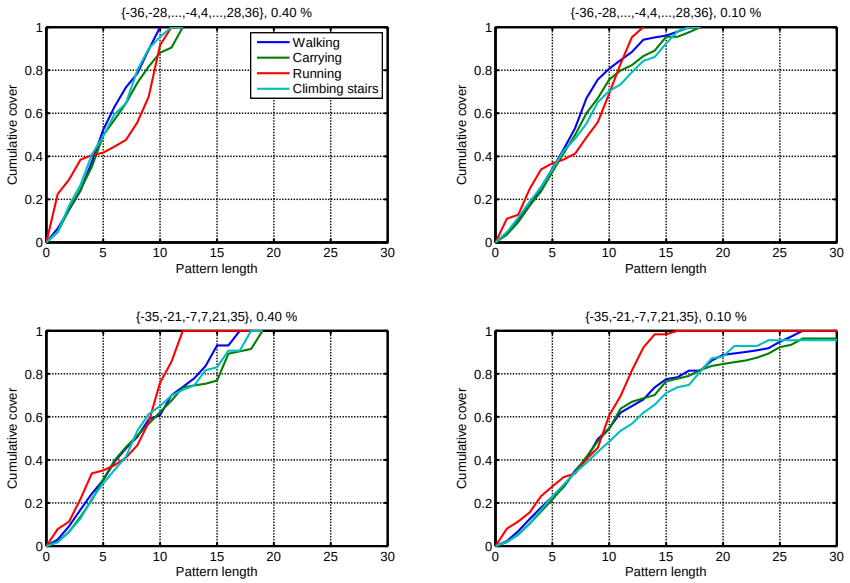

Figure 5.2: Cumulative cover vs. length of patterns under variation of breakpoints and minimum support.

5.4 Variations

We investigate three variations of the results in Section 5.3. First, in Section 5.4.1, we consider the effect of using input data from ten subjects instead of five subjects. In Section 5.4.2, we look at reducing the set of frequent patterns to speed up the learning phase of *Krimp*. Finally, we look at the classification performance when we add *cycling* and *sitting* to the data set in Section 5.4.3.

5.4.1 Ten Subjects

We expect increase in performance because a better generalization is achieved when using data from more subjects. Consider the classification results for using 10 subjects compared to 5 subjects in Table 5.9. For each choice of breakpoints and minimum support more subjects gives a better result. The increase in accuracy ranges from 3.2% to 5.6%. From this we make the important conclusion the differences between people are too big for *Krimp* to find good generalizations given sample data from only a few persons.

Most probably the results could be improved further by using data from even more subjects or lowering the support. Unfortunately a test with cross-validation, breakpoint settings 6, minimum support 0.20% and data from 10 subjects already takes 80 hours with our current implementation in Matlab on a PC with a 1.86 GHz Intel Core 2 Duo Processor (only one core is used by Matlab). So we leave this challenges open for others.

Breakpoints	Minsup	Accuracy 10 Subjects	Accuracy 5 Subjects
(1) $\{-34,-30,\ldots,-2,$ $2,\ldots,30,34\}$	0.40%	75.3%	70.8%
	0.20%	76.9%	73.2%
(2) $\{-33,-27,\ldots,-3,$ $3,\ldots,27,33\}$	0.40%	75.2%	71.5%
	0.20%	77.3%	72.8%
(3) $\{-36,-28,\ldots,-4,$ $4,\ldots,28,36\}$	0.40%	75.8%	72.6%
	0.20%	78.2%	74.5%
(4) $\{-35,-25,\ldots,-5,$ $5,\ldots,25,35\}$	0.40%	74.8%	70.4%
	0.20%	76.7%	71.1%
(5) $\{-30,-18,-6,$ $6,18,30\}$	0.40%	69.2%	64.4%
	0.20%	75.7%	70.8%
(6) $\{-35,-21,-7,$ $7,21,35\}$	0.40%	71.2%	69.5%
	0.20%	74.9%	69.3%

Table 5.9: Overview of all classification results using subjects 1 to 10.

5.4.2 Reducing The Frequent Pattern Set

In Section 4.2.1 we explained reducing the set of frequent patterns reduces the computational complexity of the learning phase of *Krimp*, which accounts for $>$ 99.5% of the total computational cost of a test run. In Table 5.10 are the results of the classification using only maximally frequent patterns. The classification accuracy is not easy to compare, in some cases maximally frequent patterns performs better in other cases all frequent patterns performs better. For breakpoint settings 5 and 6, which perform the best overall, all frequent patterns is in general superior at equal settings of *minsup*. An important trend we observe in the results is the best accuracy given fixed breakpoints is never achieved with the lowest minimum support. We conclude for the purpose of classification the description of the data becomes worse when using maximally frequent patterns with support lower than $0.1 \sim 0.05\%$.

The speed increase is significant. In Figure 5.3 we find a comparison of the number of frequent patterns generated as candidates for the Code Tables and the number of patterns selected for the Code Tables. We discuss only the cases with low minimum support (0.10%, 0.05%). The number of frequent patterns generated are two to six times lower in the maximally frequent pattern case, depending on the choice of breakpoints. We observe the number of maximally frequent patterns at each level of *minsup* is quite stable for different breakpoint settings, and is actually a bit smaller for increased interval sizes, whereas in the all frequent patterns case the number of frequent patterns grows rapidly. For low minimum support the number of patterns selected for the Code Tables is bigger than when using all frequent patterns.

Considering these overviews we conclude using maximally frequent patterns is not advisable, because the only positive aspect is computational costs of training and the negative aspect is worse performance and bigger Code Tables. Training the classifier needs to be done only once and after that the computational cost of the classifier is important, which depends only on the Code Table size.

Breakpoints	Minsup	Accuracy All Freq Patterns	Accuracy Max Freq Patterns
(1) $\{-34,-30,\ldots,-2,$ $2,\ldots,30,34\}$	1.600%	71.1%	73.2%
	0.800%	71.7%	72.4%
	0.400%	70.8%	65.9%
	0.200%	73.2%	71.9%
	0.100%	74.1%	75.6%
	0.050%	75.6%	72.4%
	0.025%		68.3%
(2) $\{-33,-27,\ldots,-3,$ $3,\ldots,27,33\}$	1.600%	67.0%	70.4%
	0.800%	67.4%	68.5%
	0.400%	71.5%	65.7%
	0.200%	72.8%	72.4%
	0.100%	75.4%	73.7%
	0.050%	74.3%	73.0%
	0.025%		72.8%
(3) $\{-36,-28,\ldots,-4,$ $4,\ldots,28,36\}$	1.600%	73.4%	74.5%
	0.800%	69.1%	68.7%
	0.400%	72.6%	68.9%
	0.200%	74.5%	72.6%
	0.100%	75.6%	72.4%
	0.050%	75.2%	73.0%
	0.025%		70.2%
(4) $\{-35,-25,\ldots,-5,$ $5,\ldots,25,35\}$	1.600%	71.3%	69.1%
	0.800%	64.1%	61.3%
	0.400%	70.4%	64.4%
	0.200%	71.1%	69.8%
	0.100%	72.8%	71.7%
	0.050%	75.4%	73.7%
	0.025%		72.8%
(5) $\{-30,-18,-6,$ $6,18,30\}$	1.600%	61.3%	62.9%
	0.800%	55.1%	62.2%
	0.400%	64.4%	56.8%
	0.200%	70.8%	66.5%
	0.100%	76.9%	74.1%
	0.050%	78.0%	74.1%
	0.025%		73.2%
(6) $\{-35,-21,-7,$ $7,21,35\}$	1.600%	67.6%	66.7%
	0.800%	67.8%	68.5%
	0.400%	69.5%	54.4%
	0.200%	69.3%	65.9%
	0.100%	74.1%	72.1%
	0.050%	80.1%	70.8%
	0.025%		69.8%

Table 5.10: Overview of all classification results using only maximal frequent patterns.

5.4.3 Cycling And Sitting

To show the maximal performance of the current solution we add to the data set the activities *cycling* and *sitting*, which should be easily separable from the other activities since they are both from another intensity category. Look at the results in Table 5.11. Both precision and recall for *cycling* and *sitting* are

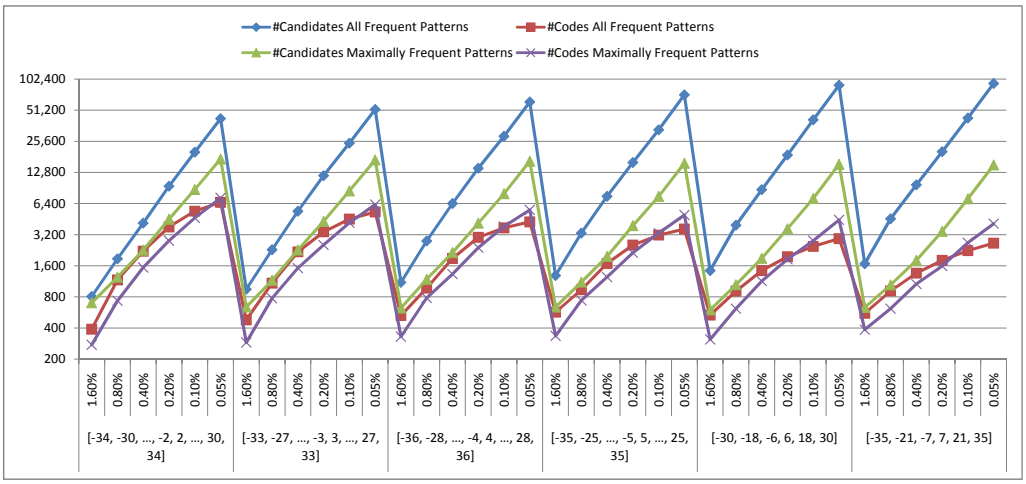

Figure 5.3: Size of maximal frequent pattern sets and Code Tables for each of the parameter settings.

above 85% and the activities are separated clearly from the other activities. An overall accuracy of 82.5% shows again the potential of *Krimp* for activity classification.

Parameter	Value						
Breakpoints	$[-\infty, -36, -28, \ldots -4, 4, \ldots, 28, 36, \infty]$						
Minimum support	0.05%						
Class category	a	b	c	i	j	r	Recall %
(a) Walking	112	41	0	0	3	5	69.6
(b) Carrying	40	130	0	0	4	1	74.3
(c) Sitting	0	0	169	0	10	0	94.4
(i) Running	0	0	0	70	0	0	100.0
(j) Cycling	2	1	0	0	116	1	96.7
(r) Climbing stairs	10	15	0	0	0	32	56.1
Precision %	68.3	69.5	100.0	100.0	87.2	82.1	

Table 5.11: Results for six activities from *Krimp*.

5.5 Detecting Anomalies

Recall the basic principle of *Krimp*. We expect *good* compression with data belonging to a certain class and *bad* compression with data not belonging to that class. In practice is it often unclear whether the compression value $\ell_{CT_i}(T)$, calculated during classification for each class i, is good or bad. In this section

we extract information from situations where compression is *reasonable*, but notably different from *good*. In particular we look at how the compression values vary for an anomalous walking pattern.

We conducted the following experiment. Six subjects wearing a triaxial accelerometer in the pocket of their pants had to walk a small predetermined course through our building. All subjects are colleagues of us. We assume the data is fairly realistic, because each subject walked at his/her own pace and in it's natural environment. Each subject had to walk down a hallway for approximately a minute, descend two floors over a few stairs, walk through a hallway again and ascend two floors over stairs where they arrived at the starting point. In total the experiments lasts about three minutes for each subject. The triaxial accelerometer is sampled at 40 Hz and we use almost the same preprocessing as discussed in Chapter 4, so normalization over 0.5 second window and smoothing by averaging over 5 values. Breakpoints are set in the same fashion by using fixed size intervals. Subject 6 had a bruised foot and used a crutch to support himself while walking and climbing stairs. Our goal is to detect this from the compression cost in *Krimp*.

Let us first look at the result of the classification from *Krimp* in Table 5.12. The minimum support has been set at 0.05% and all results are generated by using leave-one-subject-out cross-validation.

Subject	Class	1	2	Accuracy
1	(1) Walking	17	0	87.5 %
	(2) Climbing stairs	3	4	
2	(1) Walking	16	2	92.3 %
	(2) Climbing stairs	0	8	
3	(1) Walking	16	0	91.7 %
	(2) Climbing stairs	2	6	
4	(1) Walking	12	5	80.0 %
	(2) Climbing stairs	0	8	
5	(1) Walking	16	0	91.3 %
	(2) Climbing stairs	2	5	
6	(1) Walking	0	23	23.3 %
	(2) Climbing stairs	0	7	
Total	(1) Walking	77	30	75.7 %
	(2) Climbing stairs	7	38	

Table 5.12: Classification results when using cross-validation and data from subjects 1 to 6.

The overall accuracy is fairly high, and 23 out of 34 errors are made in the test set of subject 6. This makes the classification results for subject 6 indeed very different from the others. More important to recognize the anomaly are the actual compression values. First we look into another issue. We expected no errors in the "ordinary" subjects, and these errors are problematic because in order to detect differences we have to make sure our description of the "normal" case is as perfect as possible. The errors may be caused by the presence of the data of subject 6 in the learning set.

We remove subject 6 from the data set and validate the classifier again.

The result is listed in Table 5.13. It appears subject 4 now scores very bad. After evaluation it was discovered he wore the device in the back pocket of his trouser. There is a clear difference in score and apparently the patterns found by *Krimp* are not general enough to bridge the difference. It may be this issue can be resolved by using a larger data set for training, unfortunately we have no means to investigate this.

Subject	Class	1	2	Accuracy
1	(1) Walking	17	0	91.7 %
	(2) Climbing stairs	2	5	
2	(1) Walking	18	0	92.3 %
	(2) Climbing stairs	2	6	
3	(1) Walking	14	2	91.7 %
	(2) Climbing stairs	0	8	
4	(1) Walking	3	14	44.0 %
	(2) Climbing stairs	0	8	
5	(1) Walking	16	0	100.0 %
	(2) Climbing stairs	0	7	
Total	(1) Walking	68	16	83.6 %
	(2) Climbing stairs	4	34	

Table 5.13: Classification results when using cross-validation and data from subjects 1 to 5.

Subject	Class	1	2	Accuracy
1	(1) Walking	17	0	95.8 %
	(2) Climbing stairs	1	6	
2	(1) Walking	18	0	96.2 %
	(2) Climbing stairs	1	7	
3	(1) Walking	15	1	95.8 %
	(2) Climbing stairs	0	8	
5	(1) Walking	16	0	100.0 %
	(2) Climbing stairs	0	7	
Total	(1) Walking	66	1	96.9 %
	(2) Climbing stairs	2	28	

Table 5.14: Classification results when using cross-validation and data from subjects 1, 2, 3 and 5.

The results of the classification without subjects 4 and 6 are in Table 5.14. This result is indeed more satisfactory. We conclude *Krimp* has again shown it's strength in classification. In Table 5.15 we find the classification results for subjects 4 and 6 using the classifier trained on the 'normal' subjects 1, 2, 3 and 5. In the classification results there is a clear difference between the 'normal' and 'abnormal' subjects.

We proceed to the overview of compression values in Figure 5.4. On the vertical axis is the compression value for *climbing stairs* and on the horizontal axis is the compression value for *walking*. Any observation is assigned the

Subject	Class	1	2	Accuracy
4	(1) Walking	3	14	44.0 %
	(2) Climbing stairs	0	8	
6	(1) Walking	0	23	20.0 %
	(2) Climbing stairs	1	6	

Table 5.15: Classification results when trained on data from subjects 1, 2, 3 and 5.

class with the lowest compression value, thus any observation above the blue line is classified as *walking* and below the blue line as *climbing stairs*. The difference between the compression values for each class is an indicator for the certainty of an observation matching one class better than the other class. Thus observations further away from the blue line are with more certainty classified as *walking* or *climbing stairs*. All observations drawn with a triangle are samples from *walking* and all observations drawn with a circle are samples from *climbing stairs*. We can see the misclassified observations from subjects 1, 2, 3 and 5 (1 green triangle, 1 blue circle, 1 red circle) are very close to the blue line, as well as several correctly classified observations. It is also clear observations for both *walking* and *combing stairs* from subject 6 can be recognized as different from observations from subject 1, 2, 3 and 5 when considering both compression values at once. The same applies for most of the observations from subject 4.

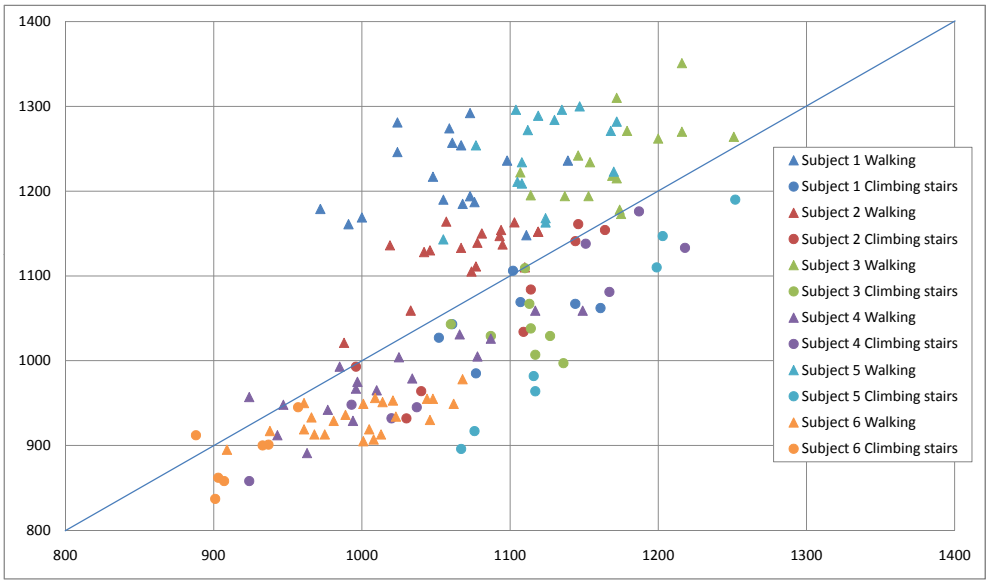

Figure 5.4: Overview of compression values for detecting anomalous data from subjects 4 and 6.

We expect good compression for observations belonging to a certain class

and bad compression for observations belonging to a different class. Remarkably this rule appears not to apply here. We know the observations from subjects 4 and 6 are different from the samples we used to train *Krimp*, but the compression is actually better for both classes. It is possible to explain this. One or more patterns that occurs frequently in the data used to train *Krimp* occurs even more often in the observations from subjects 4 and 6. Overpresence of the more frequent patterns makes the score lower instead of higher. Certainly this is unexpected and a possible solution is discussed in Section 6.1. The other problem with detecting anomalies shown in Figure 5.4 is the observations for subjects 4 and 6 are not assigned to the correct activity. So if we detect an observation to be an anomaly we do not know which activity it belongs to. We conclude it is possible to detect anomalies, but it is not as simple as we expected.

5.6 Flex Matching

The result of Table 5.14, with an overall accuracy of 96.9% is very nice, but as expressed earlier we expected no flaws at all and a possible solution may be to gather more data from more subjects. To mitigate minor differences in the acceleration pattern we introduce *flex*ible *matching* in *Krimp*. As with SAX, the symbolic representation used in this chapter lower bounds the Euclidean distance between observations. In Section 3.5 we already discussed matching patterns with a lower bound distance of zero, i.e. matching neighbouring characters such as a and b. To introduce this into *Krimp* we adjust the coding scheme and the algorithm for generating frequent patterns. The only change to the coding scheme is the definition of when a pattern *occurs* in T.

Definition 11. A pattern $p = \{p_1, \ldots, p_m\}$ *occurs* in series $T = t_1, \ldots, t_n$ at position i if and only if

$$\forall j \in \{1, \ldots, m\} : p_j + 1 \geq t_{i+j-1} \text{ and } p_j - 1 \leq t_{i+j-1}$$

So the pattern $\{a, e, b\}$ matches $\{a, e, a\}$ as well as $\{b, d, c\}$ etc. We set no maximum to the number of characters in the code that are substituted for a neighbor. The basic encoding principle remains the same. So take the last code in the code table, encode all parts of T where this code occurs and recursively encode the rest of T with the other codes in the code table. The change for the frequent pattern generation is the about the same. For each pattern we adjust the frequency count ($nontrivialSize\,(I(\{p_1, \ldots, p_m, i\}))$) to account for matches in T with characters substituted by neighboring characters.

There is a negative aspect of flex matching. The number of frequent patterns is much higher than without flex matching. In fact for each pattern p there are $3^{length(p)}$ patterns in the flex matching. Since the running time of *Krimp* grows linearly with the number of frequent patterns, an exponential growth of the number of frequent patterns cannot be coped with. To avoid generating a lot of non-interesting patterns we use the following heuristic. While extending a pattern in the algorithm, if the support of an extended pattern counting only matches where the extension-character matches exactly is less than $\frac{1}{3}$ of the support of the pattern where the extension-character matches flexibly, we do not use that character for extension. We illustrate this

with an example. Suppose we are extending frequent pattern $\{a\}$ with character b and the pattern $\{a,a\}$ occurs 20 times, $\{a,b\}$ occurs 22 times, $\{a,c\}$ occurs 28 times and $\{a,d\}$ occurs 10 times, all using flex matching. It appears pattern $\{a,c\}$ is the strongest pattern, but it may be that it does not match any part of T. Suppose with exact matching the case is $\{a,a\}$ occurs 2 times, $\{a,b\}$ occurs 18 times, $\{a,c\}$ occurs 2 times and $\{a,d\}$ occurs 8 times. In this case we should prefer patterns $\{a,b\}$ and $\{a,d\}$ and our heuristic would leave $\{a,a\}$ and $\{a,c\}$ out of the frequent pattern list (and not try to extend them), because $2 < \frac{20}{3}$ and $2 < \frac{28}{3}$ respectively. For patterns consisting of a single character we do not use flex matching.

The result for *Krimp* with flex matching for the data set of Section 5.5 using cross-validation can be found in Table 5.16. The results for subjects 4 and 6 using the classifier trained on subjects 1, 2, 3 and 5 is in Table 5.17. In both cases we use a minimum support of 0.4% instead of 0.05% as in Section 5.5 to keep the number of frequent patterns tractable. The results for the 'normal' subjects are flawless with an accuracy of 100.0%. Another positive aspect of flex matching is the Code Table becomes smaller. The Code Table trained on subjects 1, 2, 3 and 5 contains 122 codes for *walking* and 98 codes for *climbing stairs* compared to 243 codes for *walking* and 139 codes for *climbing stairs* without flex matching. This is important for application on mobile devices because smaller Code Tables require less storage space and computational power (and thus battery power).

Subject	Class	1	2	Accuracy
1	(1) Walking	17	0	100.0 %
	(2) Climbing stairs	0	7	
2	(1) Walking	18	0	100.0 %
	(2) Climbing stairs	0	8	
3	(1) Walking	16	0	100.0 %
	(2) Climbing stairs	0	8	
5	(1) Walking	16	0	100.0 %
	(2) Climbing stairs	0	7	
Total	(1) Walking	67	0	100.0 %
	(2) Climbing stairs	0	30	

Table 5.16: Classification results when using cross-validation, *flex matching* and data from subjects 1, 2, 3 and 5.

Subject	Class	1	2	Accuracy
4	(1) Walking	4	13	32.0 %
	(2) Climbing stairs	4	4	
6	(1) Walking	1	22	26.7 %
	(2) Climbing stairs	0	7	

Table 5.17: Classification results using *flex matching* when trained on data from subjects 1, 2, 3 and 5.

Since flex matching outperforms standard *Krimp* on this data set it is in-

teresting to see whether it is easier to detect the anomalies in the data from subjects 4 and 6. Consider Figure 5.5 giving the overview of the compression values. The difference between the "normal" and the "anomalous" subjects is huge and the difference between the classes for the ordinary subjects is also clearly visible. We conclude flex matching is a clear improvement to *Krimp* for this small data set.

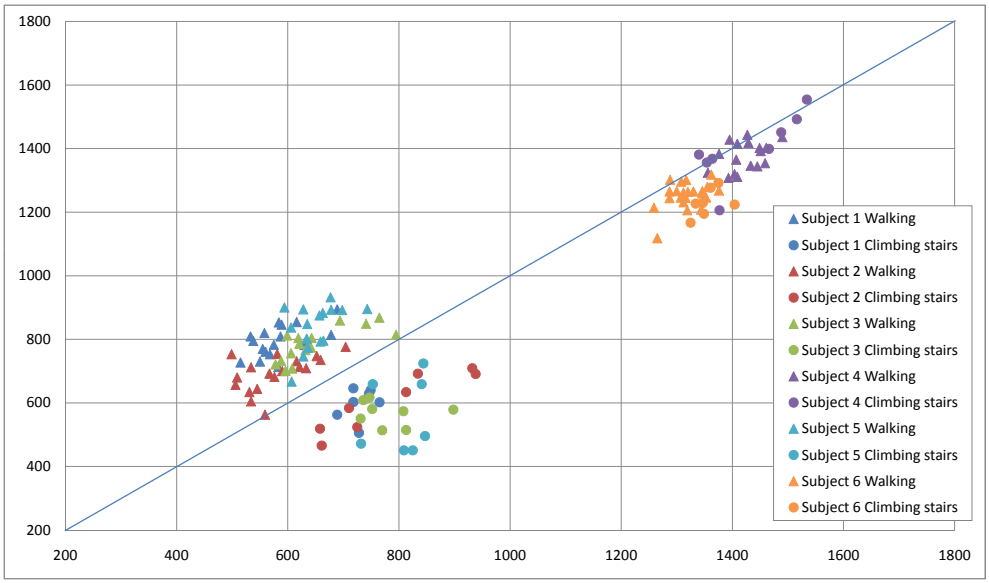

Figure 5.5: Overview of compression values for detecting anomalous data from subjects 4 and 6 using flex matching.

Chapter 6

Conclusion

In this chapter we summarize the conclusions reached throughout this thesis. We also discuss the applicability of our approach and some drawbacks in Section 6.1 and we suggest several promising directions of further research in Section 6.2.

In general, we have proved activity classification using a single on-body accelerometer can be done more robustly by using a new classification algorithm. We have created an algorithm to map numerical time-series to a discrete representation and evaluated the performance of the MDL-based classification algorithm *Krimp* on a data set for activity classification.

Before we arrived at the new classification algorithm we have shown in Chapter 2 automated cleaning of the data set is possible and advisable when working with a user-annotated data set, although we did not quantify the resulting improvement. Also, in Chapter 3 we have shown activities frequently contain patterns (motifs), which are typical for an activity and are similar between different people. Moreover these patterns could also be detected after discretization of the data set. The approach for finding motifs may be used to discover anomalous patterns, but the parameters we used are not fit to do so, as illustrated in Section 3.5.

In Chapter 5, by evaluation on a realistic publicly available data set, we have proven *Krimp* can be used to classify discretized numerical time-series. In particular it can be used to classify activities from a single wearable accelerometer. For the full set of 20 activities in the data set our approach performs worse than a standard Decision Tree (DT) with an overall accuracy of 30.9% versus 50.1% for the DT. This is caused by our restriction to an unknown sensor orientation. For some (mainly ambulatory) activities the precision and recall is much higher for *Krimp* than for DT. We have shown activities can be categorized and both the previous approach as our new algorithm make most of the recognition errors within these intensity categories. For some applications it may be useful to use the classification category instead of the specific activity because the result is more accurate. For these categorized activities our classifier outperforms DT with an accuracy of 88.5% versus 75.7% for DT. We have also evaluated the effect of the parameters of the discretization and shown the performance of the classifier does not depend a lot on these parameter choices.

In Section 5.3 we have shown our new approach can distinguish different

types of walking (*walking*, *walking while carrying*, *running*, *climbing stairs*) with up to 80.1% accuracy and when also looking at *sitting* and *cycling* the accuracy is 82.5% as can be found in Section 5.4. In Section 5.4 we have also proven the accuracy of the classifier can be improved by using data from more people than the five subjects used in all results mentioned here. Likely the results of *Krimp* still allow for improvement. In Section 5.5 we have shown anomalous behavior may be detected by using the intermediate compression values used in *Krimp*. Last but not least we have shown in Section 5.6 an adapted coding scheme for *Krimp*, flexible matching, improves the result of the classification for small data sets. Also using the improved result from flex matching we could clearly detect the data from two anomalous subjects differs from the ordinary subjects. We did not try flexible matching on the large data set, because of lack of time and because for sufficiently large data sets we do not expect a significant improvement in results.

6.1 Discussion

First we discuss two drawbacks encountered while doing the experiments described in this thesis.

1. Many machine learning algorithms have a slow learning process and *Krimp* is no exception here. During the learning phase the entire data set is encoding all over again after adding a new pattern to the Code Table. This encoding cannot be done in a linear scan over the data but requires a scan over the data for each of the codes in the Code Table in order. This makes the learning process very slow and low minimum support values unreachable for large data sets. In Section 5.4 we attempted to reduce the set of frequent patterns up front, but we did not succeed without losing accuracy of the classifier. Reducing the number of samples as is done in the PAA step of SAX may increase the speed of the process without losing accuracy.

2. Although MDL is intended to be a parameter-free learning algorithm, the combined method of applying *Krimp* to numerical time series is not parameter-free. *Krimp* itself requires only the minimum support as a parameter, which is easy to choose because lower is better. Converting numerical time series to discrete series is a more tedious process. In Section 5.3 the effect of varying breakpoints is not very big, but the precise influence on the performance remains unclear. Also the parameter choice for the normalization window (0.5 seconds) and smoothing (over 15 samples) are based on an educated guess, but not thoroughly evaluated. The choice of parameters possibly allows for improvement.

A sensor that can be worn freely in the pocket or on a belt is much more comfortable than a sensor that has to be taped on the body at a fixed location with a set orientation. However, the assumption of unknown orientation of the sensor makes detection of very low intensity activities impossible. Working on a solution for this problem is important.

In a broader sense an issue with designing algorithms for activity recognition is lack of clear goals and comparability between studies. Minnen et al.

(2006) give some solutions to standardize the performance measure of an activity classifier. Apart from the various performance measures used, a problem that remains is that each study uses a different data set, which makes results incomparable. The main reason different data sets are used is each study has a particular application in mind and there is no agreement on which activities are useful to recognize. For medical applications it would be useful to recognize activities already in use to assess independence of elderly, but no study we are aware of discusses such a set of activities. For energy-expenditure measurements it is important to recognize more vigorous and ambulatory activities and this is precisely were *Krimp* scores better than other algorithms, so in our opinion it is useful for this application.

6.2 Further Research

We have shown not all activities can be recognized by *Krimp* using a single accelerometer worn at the hip. For different types of activities different improvements could be made. To detect predominantly arm-based or upper body activities such as *brushing teeth*, *eating*, *drinking* and *folding laundry* adding an accelerometer at the wrist is probably the most promising solution. For example detection could be done at both sensors independently and compression scores could be combined by an algorithm to give a final answer. To detect postures such as *sitting* or *lying down* sensor orientation is needed. It is interesting to look for an algorithmic solution to compute this. Several activities in the data set we used, such as *riding the elevator* or *vacuum cleaning* contain no 'basic' motion which is repeated consecutively during the activity. To detect these activities and many more complex activities never studied before such as *sleeping* versus *lying down to relax* need a form of reasoning to be detected. Such activities cannot be deduced from 10 seconds of acceleration data. Such reasoning is also necessary for ambient intelligent devices, but has not received sufficient attention yet.

We discussed it is possible to detect anomalous behavior using the compression values in *Krimp* in Section 5.5. In the case without flex matching the change in values we had expected did not occur, in fact the exact opposite change occurred. Although with flex matching the difference was very clear, the same problem may have occurred. The problem has a theoretical foundation, but the solution to this problem is probably not hard. The information we should be interested in is not the compression value given by a certain Code Table, because the question whether certain data is compressed well by a Code Table cannot be answered by looking only at the compression value. This is because not all data is equally regular. The information that tells us whether data matches a Code Table is comparison of the codes distribution. If the frequent patterns occur equally often, the data sets are probably highly alike. To check this we can simply compute whether the cost $\ell_{CT}(p_i)$ associated with each code p_i is approximately equal to the cost computed when using the frequencies of each code in the new data. What is approximately equal should of course be determined taking into account the natural variance in the data set. We expect this approach works well.

6.3 Acknowledgment

It was a pleasure to write my thesis at Philips Research, for which I would like to thank all my colleagues there. I would like to express my gratitude to Ingrid Flinsenberg for giving me the opportunity to work there, for guiding me through the research and for making time for discussion and feedback whenever required.

Also I would like to thank Arno Siebes from Utrecht University for the nice discussions on available algorithms and methods and for sharing his ideas with me. Among others he suggested to use a compression-based algorithm.

For generating and sharing the extensive activity data set I thank Ling Bao and Steven Intille. Finally I thank all the participants in the self collected data set.

Bibliography

Aipperspach, R., Cohen, E. & Canny, J. (2006), Modeling human behavior from simple sensors in the home, *in* 'Proceedings Of The IEEE Conference On Pervasive Computing'.

Bao, L. (2003), Physical activity recognition from acceleration data under semi-naturalistic conditions, Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Bao, L. & Intille, S. S. (2004), Activity recognition from user-annotated acceleration data, *in* A. Ferscha & F. Mattern, eds, 'PERVASIVE 2004', Springer-Verlag, pp. 1–17.

Bouten, C. V. C., Koekkoek, K. T. M., Verduin, M., Kodde, R. & Janssen, J. D. (1997), 'A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity', *IEEE Transactions on Biomedical Engineering* **44**(3), 136–147.

Chen, J., Kwong, K., Chang, D., Luk, J. & Bajcsy, R. (2005), Wearable sensors for reliable fall detection, *in* 'IEEE International Conference of the Engineering in Medicine and Biology Society', pp. 3551–3554.

Chiu, B., Keogh, E. & Lonardi, S. (2003), Probabilistic discovery of time series motifs, *in* 'KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, pp. 493–498.

Consolvo, S., McDonald, D. W., Toscos, T., Chen, M. Y., Froehlich, J., Harrison, B., Klasnja, P., LaMarca, A., LeGrand, L., Libby, R., Smith, I. & Landay, J. A. (2008), Activity sensing in the wild: a field trial of ubifit garden, *in* 'CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems', ACM, pp. 1797–1806.

Grünwald, P. D. (2005), Minimum description length tutorial, *in* P. D. Grünwald, I. J. Myung & M. A. Pitt, eds, 'Advances in Minimum Description Length', The MIT Press.

Hayes, T. L., Pavel, M. & Kaye, J. A. (2004), An unobtrusive in-home monitoring system for detection of key motor changes preceding cognitive decline, *in* '26th Annual International Conference of the IEEE Engineering In Medicine And Biology Society', pp. 2480–2483.

House_n Research Group (2007), 'House_n web page `http://architecture.mit.edu/house_n/`'.

Huynh, T. & Schiele, B. (2006), Towards less supervision in activity recognition from wearable sensors, *in* 'IEEE International Symposium on Wearable Computers', pp. 3–10.

Intille, S. S., Bao, L., Tapia, E. M. & Rondoni, J. (2004), Acquiring in situ training data for context-aware ubiquitous computing applications, *in* 'CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, pp. 1–8.

Jain, G., Cook, D. & Jakkula, V. (2006), Monitoring health by detecting drifts and outliers for a smart environment inhabitant, *in* 'Proceedings of the International Conference On Smart Homes and Health Telematics'.

Keogh, E. (2008), 'Sax web page `http://www.cs.ucr.edu/~eamonn/SAX.htm`'.

Keogh, E. & Lin, J. (2005), 'Clustering of time-series subsequences is meaningless: implications for previous and future research', *Knowledge and Information Systems* **8**(2), 154–177.

Kirkeby, O. & Kahari, M. (2007), 'Nokia activity monitor `http://research.nokia.com/projects/activity_monitor`'.

Li, M. & Vitányi, P. (1997), *An introduction to Kolmogorov complexity and its applications (2nd ed.)*, Springer-Verlag.

Liao, L., Fox, D. & Kautz, H. (2005), Location-based activity recognition, *in* 'Advances in Neural Information Processing Systems'.

Lin, J., Keogh, E., Lonardi, S. & Chiu, B. (2003), A symbolic representation of time series, with implications for streaming algorithms, *in* 'DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery', ACM, pp. 2–11.

Logan, B., Healey, J., Philipose, M., Tapia, E. M. & Intille, S. S. (2007), A long-term evaluation of sensing modalities for activity recognition., *in* J. Krumm, G. D. Abowd, A. Seneviratne & T. Strang, eds, 'Ubicomp', Vol. 4717 of *Lecture Notes in Computer Science*, Springer, pp. 483–500.

Minnen, D., Starner, T., Ward, J. A., Lukowicz, P. & Troster, G. (2005), Recognizing and discovering human actions from on-body sensor data, *in* 'IEEE International Conference on Multimedia and Expo', pp. 1545–1548.

Minnen, D., Westeyn, T., Starner, T., Ward, J. A. & Lukowicz, P. (2006), Performance metrics and evaluation issues for continuous activity recognition, *in* 'Performance Metrics in Intelligent Systems Workshop (PerMIS)'.

Patterson, D. J., Fox, D., Kautz, H. A. & Philipose, M. (2005), Fine-grained activity recognition by aggregating abstract object usage, *in* 'IEEE International Symposium on Wearable Computers', pp. 44–51.

Siebes, A. P. J. M., Vreeken, J. & Van Leeuwen, M. (2006), Item sets that compress, *in* 'Proceedings of the ACM SIAM Conference on Data Mining', pp. 393–404.

Stikic, M., Huynh, T., Van Laerhoven, K. & Schiele, B. (2008), Adl recognition based on the combination of rfid and accelerometer sensing, *in* 'Proceedings of Pervasive Health 2008: 2nd International Conference on Pervasive Computing Technologies for Healthcare'.

Tapia, E. M., Intille, S. S., Haskell, W., Larson, K., Wright, J., King, A. & Friedman, R. (2007), Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor, *in* 'IEEE International Symposium on Wearable Computers', pp. 37–40.

Van Laerhoven, K. & Cakmakci, O. (2000), What shall we teach our pants?, *in* 'Proceedings of the 4th IEEE International Symposium on Wearable Computers', IEEE Computer Society.

Van Leeuwen, M., Vreeken, J. & Siebes, A. P. J. M. (2006), Compression picks item sets that matter, *in* J. Furnkranz, T. Scheffer & M. Spiliopoulou, eds, 'Knowledge Discovery in Databases: PKDD', Springer, pp. 585–592.

Verghese, J., Lipton, R. B., Hall, C. B., Kuslansky, G., Katz, M. J. & Buschke, H. (2002), 'Abnormality of gait as a predictor of non-alzheimer's dementia', *New England Journal Of Medicine* **347**(22), 1761–1768.

Verhaegh, W., Aarts, E. & Korst, J. (2003), *Algorithms in ambient intelligence*, Vol. 2 of *Philips Research*, Kluwer.

Wang, S., Pentney, W., Popescu, A.-M., Choudhury, T. & Philipose, M. (2007), Common sense based joint training of human activity recognizers, *in* 'Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI)'.

Witten, I. H. & Frank, E. (2005), *Data Mining: Practical machine learning tools and techniques, 2nd Edition*, Morgan Kaufmann.

Appendix A

Classification Results

Full classification results for the data set of Bao & Intille (2004) for various settings of breakpoints and minimum support. #*Cand* is the total number of all frequent patterns generated over all cross-validation runs and all classes, so $Average(\#Cand) = \#Cand/(5 \cdot 4)$ for 5 subjects and $Average(\#Cand) = \#Cand/(10 \cdot 4)$ for 10 subjects. *Select* % is the relative number of frequent patterns selected for the final Code Table. All experiments where run using Matlab on a PC with a Intel Core 2 Duo 1.86 GHz processor, but Matlab uses only a single core.

Breakpoints	Minsup	#Cand	Select %	Accuracy	Runtime
$\{-34, -30, \ldots, -2,$	1.60%	807	48.3%	71.1%	0:14:54
$2, \ldots, 30, 34\}$	0.80%	1,870	62.6%	71.7%	0:53:04
	0.40%	4,144	53.6%	70.8%	2:26:35
	0.20%	9,428	40.4%	73.2%	5:57:59
	0.10%	20,155	26.8%	74.1%	13:00:37
	0.05%	42,500	15.5%	75.6%	27:44:12
$\{-33, -27, \ldots, -3,$	1.60%	950	50.5%	67.0%	0:22:48
$3, \ldots, 27, 33\}$	0.80%	2,293	47.5%	67.4%	1:10:02
	0.40%	5,396	40.5%	71.5%	3:21:04
	0.20%	11,915	28.6%	72.8%	7:43:30
	0.10%	24,594	18.4%	75.4%	16:10:20
	0.05%	52,123	10.2%	74.3%	38:39:10
$\{-36, -28, \ldots, -4,$	1.60%	1,105	48.0%	73.4%	0:28:19
$4, \ldots, 28, 36\}$	0.80%	2,785	35.0%	69.1%	1:20:36
	0.40%	6,417	29.4%	72.6%	3:47:16
	0.20%	14,069	21.5%	74.5%	8:58:34
	0.10%	28,672	13.0%	75.6%	18:43:08
	0.05%	61,618	6.9%	75.2%	46:49:30
$\{-35, -25, \ldots, -5,$	1.60%	1,289	44.2%	71.3%	0:36:26
$5, \ldots, 25, 35\}$	0.80%	3,319	28.3%	64.1%	1:45:13
	0.40%	7,529	22.4%	70.4%	4:36:04
	0.20%	15,993	15.9%	71.1%	10:27:30
	0.10%	33,079	9.6%	72.8%	22:15:41
	0.05%	72,099	5.0%	75.4%	59:05:00
$\{-30, -18, -6,$	1.60%	1,438	37.2%	61.3%	0:44:23
$6, 18, 30\}$	0.80%	3,956	23.0%	55.1%	2:19:18
	0.40%	8,731	16.5%	64.4%	5:39:28
	0.20%	18,895	10.4%	70.8%	12:44:49
	0.10%	41,395	6.0%	76.9%	29:11:00
	0.05%	89,408	3.3%	78.0%	71:01:10
$\{-35, -21, -7,$	1.60%	1,676	33.1%	67.6%	0:56:06
$7, 21, 35\}$	0.80%	4,560	20.1%	67.8%	3:01:23
	0.40%	9,719	14.0%	69.5%	7:00:46
	0.20%	20,367	8.9%	69.3%	15:19:16
	0.10%	43,047	5.2%	74.1%	37:42:50
	0.05%	93,046	2.8%	80.1%	80:59:40

Figure A.1: Classification results for subjects 1:5 using all frequent patterns.

Breakpoints	Minsup	#Cand	Select %	Accuracy	Runtime
$\{-34, -30, \ldots, -2,$	1.60%	702	39.2%	73.2%	0:09:25
$2, \ldots, 30, 34\}$	0.80%	1,242	59.2%	72.4%	0:23:51
	0.40%	2,264	68.0%	65.9%	0:46:39
	0.20%	4,513	62.0%	71.9%	1:34:07
	0.10%	8,740	53.2%	75.6%	3:02:09
	0.05%	17,254	42.1%	72.4%	5:34:19
	0.03%	34,044	29.8%	68.3%	10:06:54
$\{-33, -27, \ldots, -3,$	1.60%	638	45.5%	70.4%	0:10:38
$3, \ldots, 27, 33\}$	0.80%	1,157	67.0%	68.5%	0:23:34
	0.40%	2,272	66.9%	65.7%	0:47:14
	0.20%	4,298	59.2%	72.4%	1:28:22
	0.10%	8,472	49.2%	73.7%	2:51:25
	0.05%	16,918	37.0%	73.0%	5:09:55
	0.03%	34,033	25.6%	72.8%	9:19:49
$\{-36, -28, \ldots, -4,$	1.60%	624	52.9%	74.5%	0:10:42
$4, \ldots, 28, 36\}$	0.80%	1,182	66.0%	68.7%	0:23:13
	0.40%	2,144	62.0%	68.9%	0:43:07
	0.20%	4,131	58.0%	72.6%	1:23:09
	0.10%	7,961	48.2%	72.4%	2:33:30
	0.05%	16,420	33.9%	73.0%	4:48:33
	0.03%	33,237	23.5%	70.2%	8:42:17
$\{-35, -25, \ldots, -5,$	1.60%	638	52.5%	69.1%	0:11:54
$5, \ldots, 25, 35\}$	0.80%	1,112	66.5%	61.3%	0:21:50
	0.40%	1,973	63.1%	64.4%	0:39:50
	0.20%	3,906	54.8%	69.8%	1:16:25
	0.10%	7,499	44.2%	71.7%	2:19:22
	0.05%	15,690	31.6%	73.7%	4:28:30
	0.03%	34,033	25.6%	72.8%	9:19:49
$\{-30, -18, -6,$	1.60%	598	51.8%	62.9%	0:10:52
$6, 18, 30\}$	0.80%	1,048	58.7%	62.2%	0:21:16
	0.40%	1,890	60.3%	56.8%	0:36:35
	0.20%	3,653	50.6%	66.5%	1:08:36
	0.10%	7,148	39.2%	74.1%	2:08:09
	0.05%	15,330	29.0%	73.2%	4:13:48
	0.03%	33,237	23.5%	70.2%	8:42:17
$\{-35, -21, -7,$	1.60%	632	60.9%	66.7%	0:12:21
$7, 21, 35\}$	0.80%	1,046	58.8%	68.5%	0:21:00
	0.40%	1,804	59.0%	54.4%	0:34:02
	0.20%	3,439	46.2%	65.9%	1:02:21
	0.10%	7,116	37.7%	72.1%	2:07:14
	0.05%	15,101	27.1%	70.8%	4:02:45
	0.03%	31,831	22.2%	69.8%	8:28:01

Figure A.2: Classification results for subjects 1:5 using only maximally frequent patterns.

Breakpoints	Minsup	#Cand	Select %	Accuracy	Runtime
$\{-34,-30,\ldots,-2,$	0.40%	8,214	69.3%	75.3%	11:56:08
$2,\ldots,30,34\}$	0.20%	18,526	54.7%	76.9%	30:10:00
$\{-33,-27,\ldots,-3,$	0.40%	10,597	52.7%	75.2%	16:27:43
$3,\ldots,27,33\}$	0.20%	23,403	40.7%	77.3%	39:28:20
$\{-36,-28,\ldots,-4,$	0.40%	12,560	35.5%	75.8%	18:03:54
$4,\ldots,28,36\}$	0.20%	27,721	28.1%	78.2%	44:10:10
$\{-35,-25,\ldots,-5,$	0.40%	14,724	26.4%	74.8%	22:09:37
$5,\ldots,25,35\}$	0.20%	31,629	21.2%	76.7%	51:37:50
$\{-30,-18,-6,$	0.40%	17,174	19.4%	69.2%	27:02:05
$6,18,30\}$	0.20%	37,234	14.2%	75.7%	62:11:50
$\{-35,-21,-7,$	0.40%	18,853	17.2%	71.2%	32:45:40
$7,21,35\}$	0.20%	39,918	11.6%	74.9%	75:40:10

Figure A.3: Classification results for subjects 1:10 using all frequent patterns.

Breakpoints	Minsup	#Cand	Select %	Accuracy	Runtime
$\{-34,-30,\ldots,-2,$	0.40%	4,571	79.9%	71.9%	4:19:02
$2,\ldots,30,34\}$	0.20%	8,965	73.6%	74.5%	8:35:42
	0.10%	17,667	65.8%	76.4%	17:59:30
	0.05%	34,114	54.5%	78.5%	27:48:20
$\{-33,-27,\ldots,-3,$	0.40%	4,577	77.3%	71.2%	4:19:34
$3,\ldots,27,33\}$	0.20%	8,736	71.5%	72.6%	9:18:38
	0.10%	17,084	62.7%	77.7%	16:38:19
	0.05%	33,994	49.1%	76.1%	26:13:00
$\{-36,-28,\ldots,-4,$	0.40%	4,288	73.7%	71.0%	3:58:00
$4,\ldots,28,36\}$	0.20%	8,329	68.3%	74.8%	7:42:58
	0.10%	16,066	59.9%	72.8%	14:58:11
	0.05%	32,642	45.3%	76.9%	24:01:23
$\{-35,-25,\ldots,-5,$	0.40%	3,929	70.2%	68.4%	3:42:32
$5,\ldots,25,35\}$	0.20%	7,958	66.1%	72.5%	7:14:33
	0.10%	15,153	55.0%	74.3%	13:35:07
	0.05%	31,116	42.2%	74.8%	21:40:29
$\{-30,-18,-6,$	0.40%	3,799	68.2%	60.8%	3:24:05
$6,18,30\}$	0.20%	7,409	59.3%	70.1%	6:49:04
	0.10%	14,394	49.6%	78.7%	12:37:20
	0.05%	30,461	37.5%	76.1%	20:12:13
$\{-35,-21,-7,$	0.40%	3,658	68.6%	54.9%	3:15:01
$7,21,35\}$	0.20%	7,025	57.4%	67.9%	6:01:17
	0.10%	14,562	47.7%	76.1%	12:27:09
	0.05%	30,076	36.3%	75.9%	19:51:02

Figure A.4: Classification results for subjects 1:10 using only maximally frequent patterns.