Machine Learning methods for mood disorder decision support

Tomasz Artur Oleksy

Master's thesis in Software Engineering at

Department of Computing, Mathematics and Physics,
Bergen University College

Department of Informatics,
University of Bergen

June 2017

Western Norway
University of
Applied Sciences

Acknowledgements

I wish to thank my supervisors, Svein Ivar Lillehaug and Yngve Lamo for their valuable guidance during the process of writing this assignment.

I would also like to thank prof. Ole Bernt Fasmer and Michael Alexander Riegler for their helpful assistance when I had burning questions.

Finally, I would like to thank Agnieszka for her support as well as the kids for showing unusual consideration by stampeding around, and not through me and my laptop.

Abstract

Early, automated recognition of symptoms is the optimal goal of any health monitoring system. In bipolar disorder, early detection of impending mood episodes would be of great value, reducing costs, leading to improved treatment and lowering consequential risks to the patient. As a first step in creating such a system, we explore the viability of using machine learning for accurate detection of specific physiological states exclusively from data collected by a wearable actigraph device.

We deal with noisy time series data posing challenges with extreme class imbalance and low positive sample counts. An unsupervised method of anomaly detection (Hierarchical Temporal Memory) followed by a selection of supervised learning methods is applied to the data to find the best performing combination for detection of attacks (positives). Optimal methods for dealing with class imbalance are selected. Furthermore, various preprocessing techniques combined with feature extraction are applied to improve classifier performance.

Due to the unforeseen absence of adequate sensor data from bipolar studies, a dataset from a migraine-study involving bipolar patients has been adapted for the experiments.

Some promising results have been obtained from hundreds of experiment runs. Much more data is needed to provide reliable conclusions, however the combination of methods used should provide a very good starting point for further attempts in this area.

Contents

Table of Abbreviations

ADL	Activities of daily life
AI	Artificial intelligence
ANN	Artificial neural network
AUC	Area Under the Curve
BMI	Body mass index
BSN	Body sensor network
CFS	Correlation-based Feature Selection
CLA	Cortical Learning Algorithm
ET	Extremely Randomized Trees
FFT	Fast Fourier Transforms
FPR	False Positive Rate
GPU	Graphical Processing Unit
HP	Hodrick-Prescott
HR	Heart rate
IBA	Index of Balanced Accuracy
KNN	K-Nearest Neighbour
LR	Logistic Regression
MCC	Matthews Correlation Coefficient
ML	Machine learning
MS	Multiple sclerosis
PCA	Principal Component Analysis
PPV	Positive Predictive Value
RFE	Recursive Feature Elimination
ROC	Receiver Operating Characteristic
SDR	Sparse Dynamic Representation
SEM	Standard error of the mean
SMOTE	Synthetic Minority Oversampling TEchnique
SOS	Simple over-sampling
SVM	Support Vector Machines
TNR	True Negative Rate
TP	Temporal Pooler
TPR	True Positive Rate
TST	Total Sleep Time
WASO	Wake after sleep onset
WBAN	Wireless body area network
WBSN	Wireless body sensor network
WHMS	Wearable health monitoring systems

1 Introduction

In this chapter, we briefly present the general underlying motivating factors in context of the healthcare system, the healthcare staff and the patients, for the work described in this document. We touch on some insights into how combination of various sensor technologies and machine learning analysis methods add value to the traditional methods of patient monitoring and patient health care.

In section 1.4 under, we present the research questions and goals for this master thesis, followed by known major limitations. Finally, an overview over the remaining chapters is given in section 1.6.

1.1 Motivation

The typical modern lifestyle brings with it increased stress levels, which, when allowed to exert effect on the human psyche over an extended period, might lead to a whole range of problematic physical and psychological symptoms. Mental disorders can partly be ascribed to this, though other factors like genetics have also been shown to have influence. Accurate diagnosis and effective treatment of mental disorders, no matter the underlying cause, is of critical importance to the individuals affected but also to their families, relatives and the society in general. The social, economic and physical consequences of non-treatment and/or insufficient treatment can be dramatic and even fatal. With this in mind, as well as the general increase in population average lifespan and the increasing healthcare costs following as a natural consequence, the need to monitor patient's health situation in an ecologically valid environment has become of ever greater value [1, 2].

A heavily weighing factor on healthcare cost is linked to the resources needed to cater for the observation and treatment of patients with chronic, persistent or recurring types of ailments. In such cases, regular consultations are typically recommended but highly trained staff and their time represents an expensive resource. Most regular consultations could be omitted if the clinician/therapist had the possibility of monitoring the patient's state without having to attend physically at scheduled times, doing routine, low-value tasks and medical check-ups [3]. Furthermore, there are other limitations: patient's own recollections are more often than not unreliable due to personal perceptual bias and inaccurate reporting [1], feelings of disengagement and potential delayed intervention from health professionals due to late detection of symptoms [4]. Early symptom detection is important for treatment. Patients suffering from specific mental disorders, such as bipolar disorder, unipolar disorder and schizophrenia that are considered incurable lifelong conditions carry a risk of critical relapses

or phase transitions [5] which should ideally be recognized as early as possible - preferably well ahead of time so that adequate actions can be undertaken.

Monitoring systems have been proposed to improve the accessibility and level of patient interaction, while still greatly depending on the cooperation of the subjects [4], however exclusive dependence on manual self-reporting still poses a disadvantage. To remedy that, accurate, independent, non-invasive monitoring methods provide a potential solution. A wide ranging and diverse spectrum of sensor technologies are being developed which present new possibilities for use in medical applications, potentially improving the daily dealings of both patients and health professionals. Rapid improvements of wireless technology have led to the development of comprehensive patient monitoring systems like wireless body area network (WBAN) and body sensor network (BSN), both proposed for effective monitoring of various types of physiological activities [6].

Proposed wearable health monitoring systems (WHMS) consisting of one or more miniaturized sensors, wearable or even implantable are capable of measuring parameters such as heart rate, blood pressure, temperature, oxygen saturation, respiration rate, galvanic skin-conductivity, motor activity, etc. Actigraphy has traditionally been used in studies of sleep patterns, as motor activity patterns have shown strong correlation with a variety of sleep-related disorders [7],[8]. The actigraph as well as its main subcomponent, the accelerometer, belongs to the group of relatively well-established, robust, simple low-cost wearable motor activity sensors also found in most smartphones today.

Currently, it is the combination of wearable sensor technologies with machine learning (ML) techniques that show the most promising results for improved healthcare (as can be seen in the background literature section). Both the simpler, already existing devices as well as the more sophisticated wireless monitoring systems [2, 6] that have been proposed in out of institution health monitoring settings, all share a common underlying need for effective running analysis of the produced sensor data. Countless studies exist on the use and adaptation of sensor data for inferencing, decision support[1] and diagnosis. We believe that it is the correct and effective interpretation of this data that can play a major role in ensuring prompt reaction in case of medical emergency as well as eliminating the lengthy process before a clinical decision is undertaken.

[1] Medical Decision support is an area belonging to the field of Health Informatics, that mainly deals with computer-aided systems for the improvement of the quality of healthcare at various stages in the treatment process by providing easy & quick contextual information at the right time to the practitioner that might need it.

1.2 Bipolar Disorder

By one definition, *bipolar disorder* (BP) is *"a brain disorder that causes unusual shifts in mood, energy, activity levels, and the ability to carry out day-to-day tasks" (NIMH, Bipolar Disorder)* [9].

BP is a chronic mood disorder characterized by alternating phases of elation (mania or the milder, hypomania) and depression (also called mood episodes). The manic states may cause over exaggerated self-confidence and even recklessness, while depressive states have the opposite effect. Both ends of the scale are problematic. Hypomania is a milder form of mania, an individual may feel good, be highly productive, and generally function well. Untreated, however hypomania may develop into severe mania or depression [9]. *Euthymia* is the term used for the neutral or normal state comparable to healthy individuals. A simplified diagram depicting the various states is shown in Fig. 1-1.

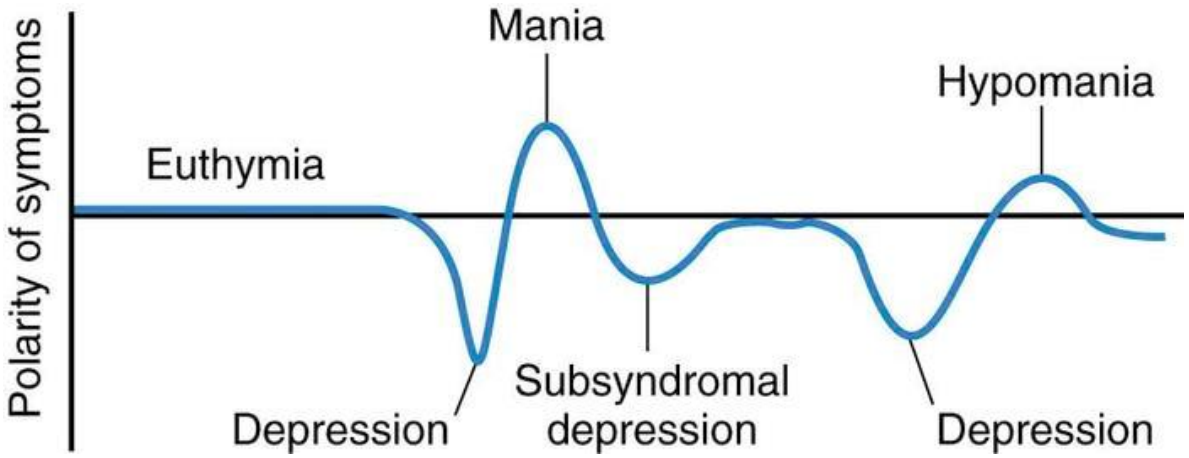

Fig. 1-1 A symbolic representation of the states in bipolar disorder. Detecting potential early changes prior to a critical state would be the ideal. Note: some states do not occur in certain types of bipolar disorder, see Table 1-1: Types of Bipolar Disorder. Based on resource from (The National Institute of Mental Health, 2017) [9]*. for details. (image source: http://neurowiki2013.wikidot.com)*

Importantly, mood changes are known to be accompanied by extreme shifts in energy, activity and behaviour [10]. Bipolar disorder is associated with devastating consequences, personal, social, and financial, but does not stop there: risk of suicide attempts increases dramatically while actual suicides occur in about 20% of cases [11]. In BP, the transitions between phases are typically few over a relatively long timeframe. Incidents can vary from 2-4 a year to more frequent, even weekly episodes, known as rapid-cycling. Several subtypes of BP have been described where the details of these characteristics vary as shown in Table 1-1.

Bipolar I	Individuals experience both depressive and manic episodes of varying lengths.
Bipolar II	Less severe manic (hypomania) episodes than BP I; depressive episodes as in BP I.
Cyclothymia	Milder, also chronic form of BP: episodes of hypomania and depression last for at least two years.
Mixed episodes	Mania and depression occur simultaneously. Individuals experience symptoms of both simultaneously.
Rapid-cycling bipolar	Patients experience at least four or more episodes within one year.

Table 1-1: Types of Bipolar Disorder. Based on resource from (The National Institute of Mental Health, 2017) [9].

Notably, at least one study found statistically significant changes in behaviour of patients up to 60 days prior to mania episodes [12] using self-reporting methods. Furthermore, relationship between moods and activity levels are known to exist and specifically bipolar order patients have been reported to have significantly lower activity levels than healthy individuals [10]. These findings could have important implications for potential early detection and *prediction* of critical mood episodes.

Prediction is a term frequently used in this work, one meaning being the widely-accepted term used for the output value of a machine learning algorithm, given some valid input.

However, the notion of *prediction of an event* should be more precisely defined as "the detection and recognition of relevant variance in the measured variables, that has been shown to precede the specific event within acceptable degree of significance". In other words, we are not stating that we can see the future but that future events have already started a process at some (physiological) sublevel, currently not reflecting in physical pain and the changes introduced by this process can be detected. We assume this general idea to be relevant to the process of state transitions in bipolar disorder patients.

1.3 Problem description

Early detection and prediction of states in patients suffering from bipolar disorder and other mental disorders is a key factor for success in the treatment of such disorders. Currently, a patient is monitored through regular contact and conversation with the therapeutic staff or through self-reporting. This can become both expensive and time consuming but could be improved by improving the monitoring methods. An improved system would be able to monitor

the patients' physiological states by means of wearable sensors and communicate health critical information.

The INTROMAT project (INtroducing personalized TReatment Of Mental health problems using Adaptive Technology), officially launched autumn 2016, is appointed by The Norwegian Research Council as one of three projects chosen in their IKTPLUSS Lighthouse call. The goal is to improve public mental health with innovative ICT solutions [13]. As a part of INTROMAT, a study will be undertaken ecologically monitoring bipolar patients over long periods, collecting sensor and voice data, including activity data from wearable actigraph devices. The collected data will be transcribed with timestamped labels of observed states of the patients. Exact details are not known to us at the time of writing.

One objective is to devise an effective, low-cost, reliable and unobtrusive early warning system capable of monitoring and prediction of impending critical mood episodes, such as mania or depression. To achieve this, the sensor data must be analysed and processed in a continuous fashion, extracting relevant information and providing decision support functionality.

1.4 Goals and Research Questions

In this thesis, we set the following goals and research questions:

Q1: Is it feasible to detect and identify actigraph patterns related to migraine attacks by using existing supervised machine learning techniques?

Q2: Is it feasible to detect actigraph patterns related to migraine attacks by using HTM anomaly detection based on historic activity data?

G1: Find the optimal combination of preprocessing method, learning algorithm, model parameters and features that may be used in a future monitoring application.

G2: Perform a comparison of selected machine learning algorithms performance based on a range of metrics specific to an imbalanced data problem.

1.5 Limitations

The original plan for this project was based on data from a large scale (N≈100) bipolar patient study which was planned to run parallel to it. The study would extend over several months up to a year and would use wearable actigraph watches worn by bipolar disorder patients to log their activity through the whole period. One of the objectives was to study potential relationships between the various phases of bipolar disorder and activity patterns. Events of interest would be labelled in the data and the original goal was to do a preliminary machine learning analysis of this data. However, neither the study nor the data has materialized in time, and consequently a decision had to be taken to use an alternative dataset that fulfilled the minimal basic requirements for the planned machine learning analysis while somehow also being related to the domain. None of the available data supplied covered transitions between states, labelled or otherwise.

For this reason, alternative datasets of bipolar disorder patients with manually recorded migraine attacks were used for the experiments. The underlying assumption is that the potential bipolar state transition events that may be recorded in the planned experiments though not directly related migraine attacks, would be handled using similar machine learning techniques.

1.6 Overview

The rest of this thesis presents the work done on actigraph sensor readings using a variety of ML analysis methods in the context of detection, classification and possibly prediction of adverse events as depicted by an actigraph. A selection of supervised and unsupervised methods is applied. *Hierarchical Temporal Memory* (HTM) is an exciting new approach to *artificial intelligence* (AI) and temporal pattern recognition which could be applied to tasks of real-time analysis and monitoring of data streams [14]. For supervised learning, we choose among well-established classification algorithms based on decision trees, boosting and ensembles as well as Support Vector Machines (SVM), K-Nearest Neighbour (KNN), Artificial Neural Networks (ANN) and others. Furthermore, we present methods of dealing with the challenges posed by strongly imbalanced datasets.

In the next chapter, (chapter 2), we inspect the migraine attacks dataset. We explore aspects which have bearing on the outcome of the experiments.

In the *Background* chapter (chapter 3), we introduce machine learning and background literature deemed as relevant to this work. We briefly introduce ML algorithms, feature extraction and selection process, evaluation methods and metrics. An introduction to the class imbalance problem and ways of dealing with it is also presented. The related studies and literature section aims to present various relevant research papers reviewed in the context of

this work. We look at actigraph studies specifically linked with bipolar disorder or migraine, followed by studies describing methods and notable results from using statistical machine learning in combination with actigraphy.

The *Methods* chapter (chapter 4) describes the general structure of the experiments, common steps, preprocessing and details of feature extraction performed specifically for this task.

Chapter 5, *Experiments and Results*, presents the experiments and their respective results. The chapter is further divided into the supervised and unsupervised sections. It is here that we provide the details and any specific requirements for the individual experiments closely followed by the result summaries and diagrams.

We discuss the results obtained in chapter 6, *Discussion*. Here, we summarize the findings of the experiments and their implications, how they impact the goals and research questions we have set in the previous chapters. This is also the section where we touch on the problems encountered as well as potential future improvements that can be made. Finally, a *Conclusion* summary is presented in chapter 7.

2 Data exploration

In this section, we describe and explore the migraine datasets supplied for use in our experiments.

The requirements for acceptance of this data for use in our experiments were that the recordings are properly annotated with accurate timestamped labels for the migraine attack event, that monitoring was continuous and that it lasted over a maximally extended period. The first requirement is necessary for supervised learning, the two remaining ones are important for the HTM algorithm. Even though potential bipolar phase transitions are absented from this data, the migraine data contains its own type of transitions related to the activity patterns before the attack vs the period during and after the attack (see below). Our assumption is that if we can detect or even predict a migraine attack then the techniques used will possibly be similar to detecting early signs of coming mood episodes in bipolar disorder patients in the same type of sensor data.

The migraine dataset comes from an earlier, small pilot study involving bipolar patients but with focus on migraine attacks. As mentioned, the study [87] attempted to find relationships between the periods 2 hours prior to, 2 hours during and 2 hours after the attack subsided. Several significant differences were found between the two first periods: mean activity, variance, sample entropy and skewness. The migraine data ranges over a period of 14 days per individual. It was collected by a wearable actigraph worn by the patients continuously day and night. The epoch or intervals are of 1 minute, and the reported value is the aggregate sum of the recorded activity during this period. In total, 29 migraine attacks are available from 4 patients, all suffering from bipolar disorder (depression state). Additionally, the patients were reported as receiving medication (mood stabilizers).

Migraine attack dataset overview

	0	1	2	3	4
count	22147.0	25910.0	31485.0	23244.0	24536.0
mean	151.44	221.35	202.62	146.95	293.43
std	273.99	347.34	357.37	294.59	408.76
min	0.0	0.0	0.0	0.0	0.0
25%	0.0	0.0	0.0	0.0	0.0
50%	13.0	36.0	0.0	9.0	131.0
75%	190.0	332.0	283.0	172.0	421.0
max	3622.0	4859.0	5931.0	3526.0	5750.0

Table 2-1 Overview of basic statistics of migraine datasets used in the experiments. (statitics shown: count of data points, mean, standard deviation, minimum, 3 quantiles and maximum value). Column id 0-3 shows four migraine patients. Column id.4 is one of the controls (with no migraine attacks) that was used in the original pilot study by Fasmer et.al.[70]

The basic statistics for the actual datasets are summarized in Table 2-1. The control included in the table (column id.4) is a healthy individual (not diagnosed with bipolar disorder) with no history of migraine attacks. Note: the data was unprocessed at this stage, some datasets were additionally modified at a later stage.

Autocorrelation (ACF) also known as *temporal correlation* [88], contains information measures of how well a time series correlates with itself over time. More specifically it is the linear dependence of a variable with itself at two points in time [89]. We are interested in finding out to what degree a sequence of values is dependent on some previous values and over what the time lag. A time series with high autocorrelation over a time period could be expected to be easier to model than one with low correlation. Crucially, time series with values not exceeding a certain confidence interval boundary (computed by the ACF-function) are considered random noise and consequently, the best method to predict a random signal is to predict its mean. Fig. 2-1 presents sample ACF plots from the migraine data. We observe insignificant autocorrelation, which may be interpreted that consecutive timestamps show little to no interdependency.

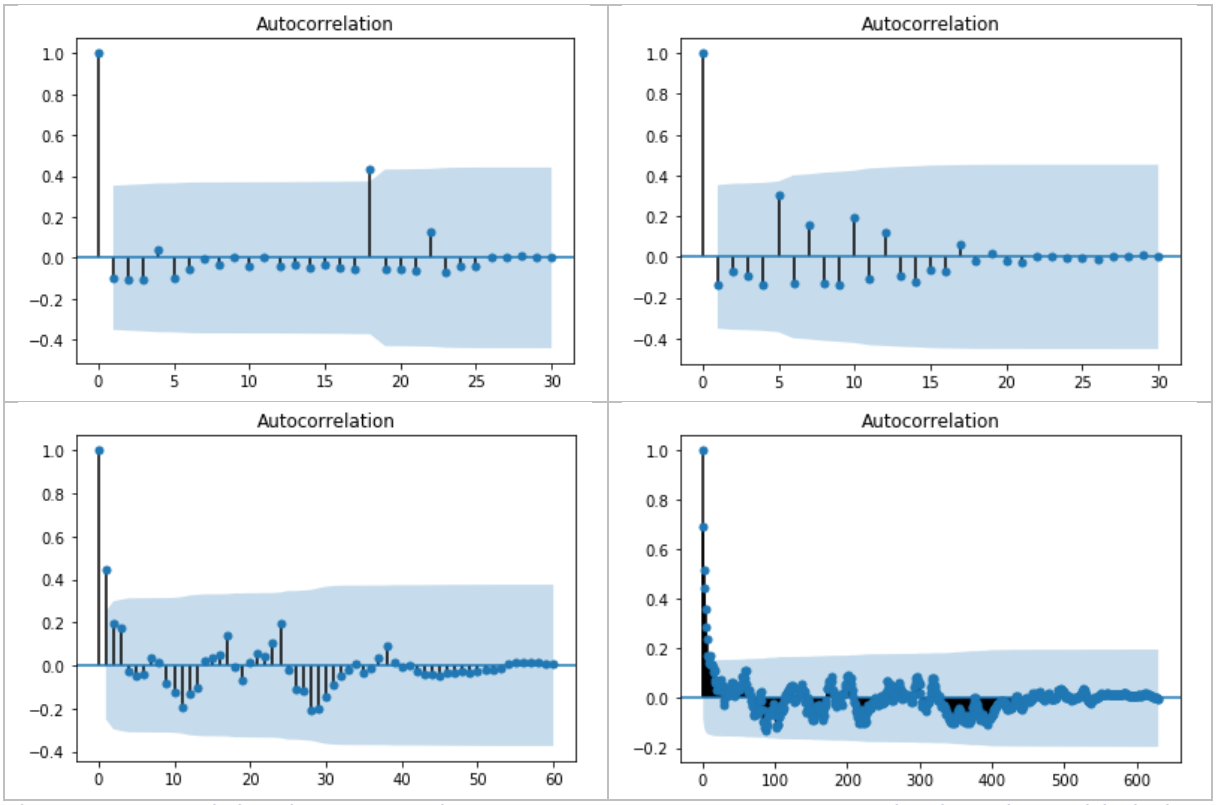

Fig. 2-1: Auto-correlation plots, Top row: lag=30, at 11.00-11.30am on two consecutive days when activity in the underlying signal is relatively high. Bottom left: lag=60, 11.00-12.00am different day. Bottom right: about 10hrs of activity from 11.00am to 21.00pm. Blue area depicts the confidence boundary, results inside basically mean noise, i.e. no significant correlation at this lag (depicted by the x-axis).

In general, on closer inspection and preliminary decomposition (not shown) of the time series sequences in the data we find a non-stationary, noisy signal with no clear, distinctive trend. The only seasonality is visible in context of day-night cycles, when activity levels are increased

during the days and decreased during the nights. Next, we take a closer look at the actual attack time sequences.

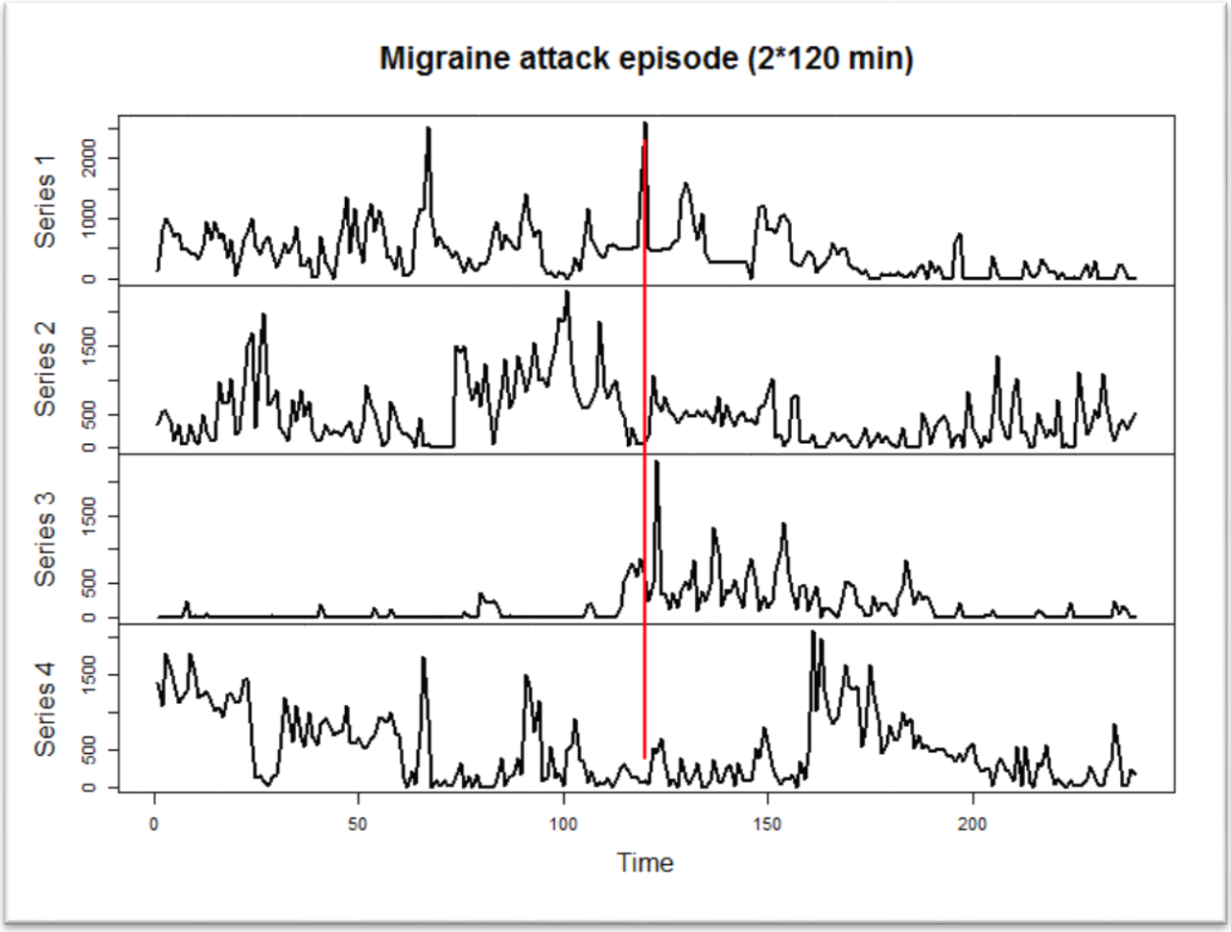

Fig. 2-2: Migraine attack episodes for one patient covering 240 minutes covering 4 separate episodes. Red line depicts time point of each attack, splitting the sequence into 2 segments: before and during.

The visual inspection of the migraine attack episodes, within a margin window of 120 minutes before and after the timestamp is show in Fig. 2-2. This is an example showing all migraine attacks from the same patient reported over a total period of the two weeks. The red line denotes where the event has been recorded. Notice the visibly apparent low correlation in the 4 episodes. Even for a human classifying these 4 sequences (or parts of them) into one common category would be a challenge. The actual correlation matrix for these sequences has been calculated for the 2*120min instances described above and is shown in Fig. 2-3. Results generally lie in the range: $-0.4 \leq x \leq 0.4$, which indicates that the correlation (as well as inverse correlation) is low to very low. This adds support to the original impression from visual inspection that these sequences have few, readily distinguishable common traits.

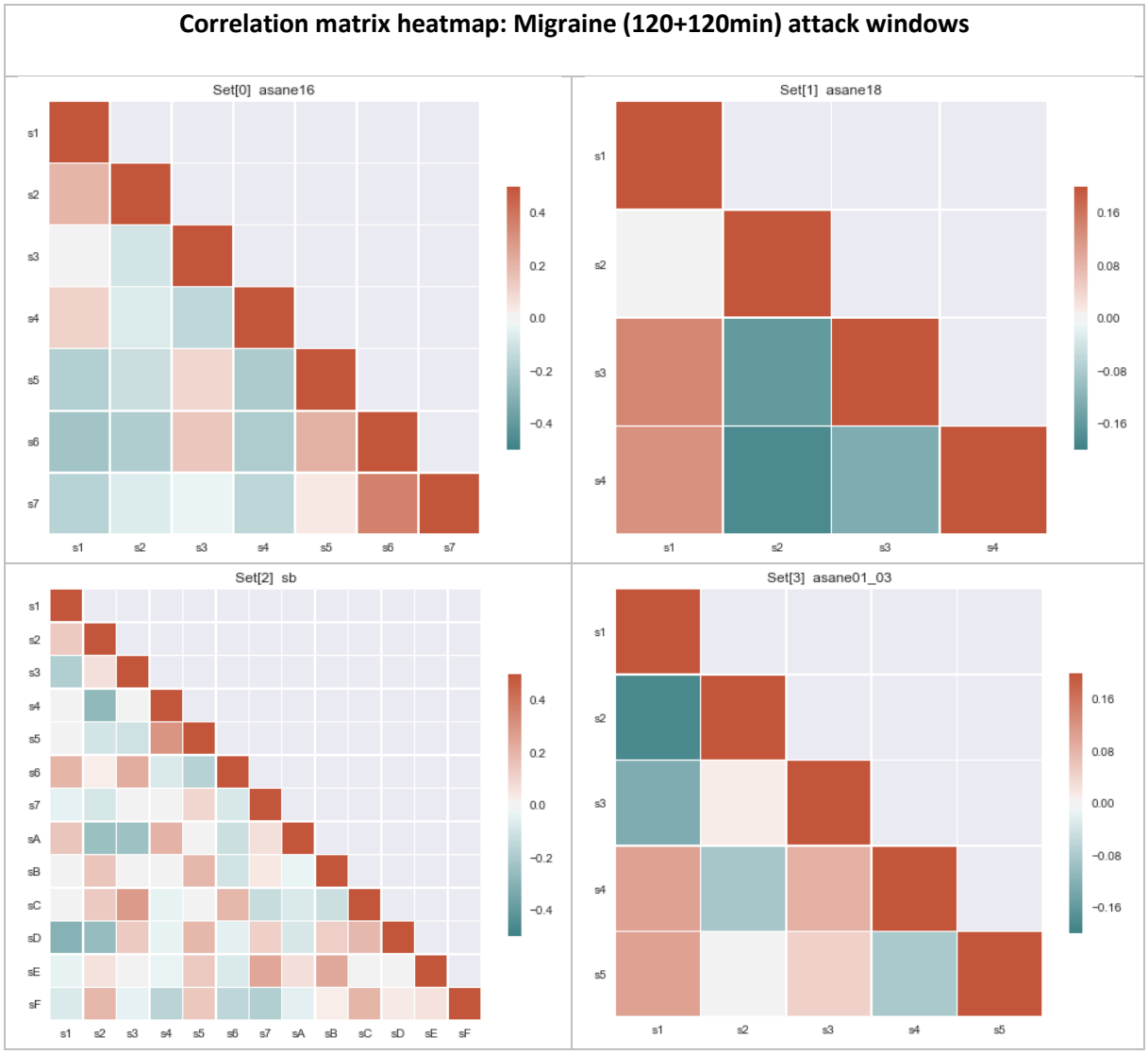

*Fig. 2-3: Heatmap matrix of correlations between all the migraine attack instances for each of the four patients. The sequences included here are all 2*120min sequences centred around the starting time of attack as reported by the patient. NB: notice the scaling is adapted for each individual dataset based on the maximum correlation found not including self-correlation. (The diagonal shows the sequences' correlation with itself, i.e. value of 1.0.)*

Even more variation can be observed in similar comparisons across different individuals. This suggests that models might need to be trained on a per individual (patient) basis to obtain useful accuracy. We decided to explore this discovery further to learn if pattern matching based on correlation or distance measure similarities could be effective in context of feature extraction for the planned anomaly detection and classification experiments.

2.1 Sequence similarity

Multiple similarity measures (*euclidean*, *cosine and Dynamic Time Warping* (DTW)) were considered in further analysis. Often, time series data mining requires effective similarity comparisons, and there is increasing evidence that the classic DTW distance is the best measure in most cases [90]. Euclidean distance and its variants (such as *manhattan* or *minkowski* distance) present several drawbacks, that make their use inappropriate in certain applications: time series must be of the same length, poor handling of noise and outliers and high sensitivity to specific signal transformations such as time-warping, amplitude scaling, shifting [91]. For the following experiments, the library "fastDTW" is required. FastDTW is an approximation algorithm for dynamic time warping, which has been shown to be one of the most accurate implementations of this type. Dynamic time warping is reportedly frequently used in speech recognition to determine if two sampled waveforms represent the same phrase [92]. We attempted to find out if sequence matching using DTW is a viable option in light of the correlation results described previously.

We performed a comparison of target sequences, using a search implementing the fastDTW library. A query sequence window makes up 15 mins before +105 mins after attack start (120 mins total). For each of the target queries selected, the distances to all remaining target sequences were calculated (Table 2-2).

2005-03-08 21:49:00	0,00	17980,00	14569,00	11496,00	17677,00	26379,00	23333.0
2005-03-12 18:15:00	17980,00	0,00	21793,00	16680,00	22356,00	24816,00	20496.0
2005-03-15 07:45:00	14569,00	21793,00	0,00	9024,00	10652,00	19302,00	12117.0
2005-03-17 22:45:00	11496,00	16680,00	9024,00	0,00	15786,00	23188,00	16012.0
2005-03-18 06:29:00	17677,00	22356,00	10652,00	15786,00	0,00	17495,00	13671.0
2005-03-19 12:00:00	26379,00	24816,00	19302,00	23188,00	17495,00	0,00	18954.0
2005-03-21 10:15:00	23333,00	20496,00	12117,00	16012,00	13671,00	18954,00	0.0

Table 2-2: Distance matrix relating the absolute DTW-distances (as calculated by fastDTW) between target labelled, 120-minute sequences (i.e. attacks) from "aasane18" migraine data containing 7 attacks. The index column shows the start timestamp.

For our example, we selected one query sequence (id: 2005-03-08 21:49:00) and found the average distance to other targets. Then, the query sequence was used in a search through all the data (*stride = 25, window size = 120*) and for every such segment, the DTW-distance to our query was calculated.

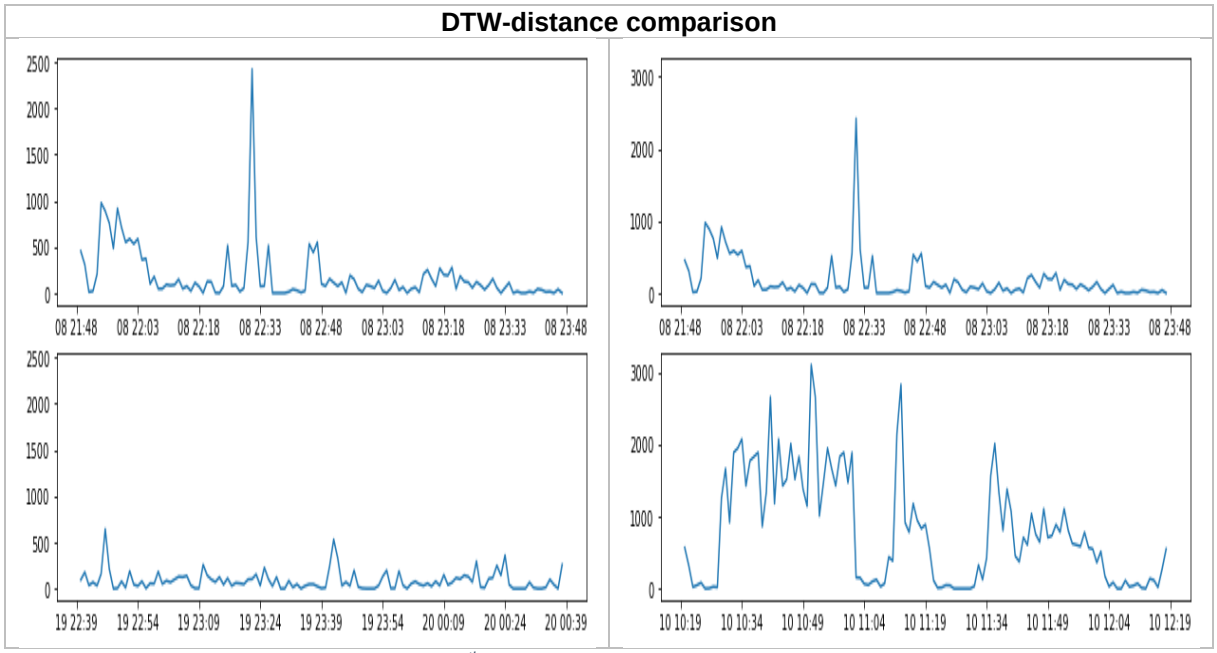

Fig. 2-4: Left column: example of good fit (5th best score, Listing 2) for query (top). Right column: example of bad fit (5th worst score) for query (top). Scores are according to fastDTW as described in DTW-experiment, 10915.0 and 50594.0 respectively.

The number of sequences where distance was less than our previously calculated average of (target) distances was reported to be 363. This is the number of other sequences in the dataset which when compared with any of the sequences in the table would on average have a better (lower) DTW-distance score. Finally, we took the minimum distance from the target table (excluding distance to self, which is 0) and get a count of how many matches had a better (i.e. lower) score than the lowest in the table. In this example, we found such 15 sequences.

```
distances.sort_values().head(15)
Out[61]:
2005-03-08 22:05:00     5170.0
2005-03-08 21:40:00     5186.0
2005-03-15 15:45:00    10505.0
2005-03-15 15:20:00    10790.0
2005-03-19 22:40:00    10915.0
2005-03-13 22:55:00    10955.0
2005-03-17 22:20:00    11093.0
2005-03-14 23:30:00    11116.0
2005-03-11 00:30:00    11199.0
2005-03-21 12:35:00    11288.0
2005-03-20 22:50:00    11422.0
2005-03-13 16:40:00    11442.0
2005-03-20 23:15:00    11452.0
2005-03-20 18:40:00    11469.0
2005-03-09 18:05:00    11487.0
dtype: float64
```

Listing 1: Lowest 15 distances to sequence query ('2005-03-08 21:49:00') in Table 2-2. Notice the 2 top entries (red) are adjacent and overlapping with the query. This is as expected. Entry #5 is plotted for comparison in Fig. 2-4.

The distance scores sorted in ascending order are shown in Listing 1. Apart from the 2 topmost entries, which are partly overlapping the query we have 13 sequences that would be more likely to be classified in the same class as the query than any of the actual target sequences from Table 2-2 given that DTW was used as distance measure. DTW is a much more accurate distance metric for comparison of sequences than pure Euclidean distance [91]. The conclusion from this experiment is that we should not expect great results from classification algorithms that use mentioned distance measures directly on sequences in the migraine data.

We explored the supplied migraine attacks dataset. It is a non-stationary, noisy time series showing no specific trend with the exception of the general day-night circadian cycle. The sequences of interest (migraine attacks) are very few and are weakly correlated. We found that even flexible distance measures like DTW as well as correlation-type of the similarity measures, cannot alone explain the class membership (i.e. attack or no attack) of these sequences. We also observed that the migraine dataset has two major challenges: the extreme class imbalance (in context of classification) and a very small positive sample count.

3 Background

This chapter is divided into two main sections. The first deals with background literature and related studies. It describes relevant aspects of working with sensor data especially in context of bipolar disorder and migraine. Results and methods of interest to this work are highlighted and the section ends with a short summary. The second section deals with necessary background on machine learning methods that were applied. This includes the complete selection of algorithms, evaluation methods and metrics that were part of our experiments.

3.1 Literature & Related studies

A large number of studies involving monitoring of mental disorders have been undertaken. From among these, the bipolar disorder and migraine related studies were specifically filtered out as the main area of interest. A comparably large amount of studies involving analysis of human activity through actigraph and accelerometer data in a variety of contexts has also been published. For this work, the emphasis was on the intersection of these two groups, i.e. where activity monitoring was studied in the context of furthering the understanding of mental disorders and related medical issues. Additionally, general studies of human *activities of daily life* (ADLs) were also included as methods and ideas therein were deemed relevant for this work.

We have no knowledge of existing studies that have attempted to solve the same (migraine attack & bipolar transitions) problem using machine learning, based solely on actigraph readings.

3.1.1 Actigraphy

Actigraphy has since late 1990's increasingly been used in circadian rhythm and quality of sleep studies, steadily taking over from *polysomnography* (PSG), which has historically been "the gold standard" of sleep assessment. Actigraphy is a useful diagnostic tool for the sleep medicine practitioner, allowing for assessment of sleep over extended periods in the natural sleep environment [8].

Furthermore, it appears to provide a valid estimate of TST (Total Sleep Time), sleep percentage and WASO (Wake after sleep onset) [67]. An actigraph is an electronic, wearable device that normally consists of a piezoelectric accelerometer with a low-pass filter (for exclusion of noise resulting from vibrations), a start/stop timer function, onboard memory for storage and hardware interface such as USB or Wi-fi for communication with computing devices. Traditionally, it is used for measuring general motor activity in order to evaluate the rest-activity cycle in study of sleep related disorders. In a basic feasibility study of actigraph utility value in neurological patient populations research, [7], supports the claim that actigraphy has gained favour as a research tool in evaluation of *circadian rhythm disorders*, *multiple sclerosis* (MS), headache and strokes. The study enforces the idea that actigraphy usage has potential in other related fields such as epilepsy and gives weight to the idea that actigraphy in combination with other tools is advantageous to more accurate results.

While it is actigraph readings that are focused on in this work, it has been suggested that combinations of different sensor types, often referred to as *sensor fusion* in literature, favourably enhance and partly validate the quality and amount of information gained from any single sensor [68].

3.1.2 Migraine & Bipolar Disorder

In the period of 2010-2015, a series of studies by Fasmer et.al. [69–72]on the relationship between healthy controls and patients diagnosed with neurological disorders such as schizophrenia and bipolar disorder, using measured activity patterns has been published.
Among these, "Objectively-measured motor activity patterns before, during and after attacks of migraine" [70], is a pilot study attempting to find relationships between periods of migraine attacks in patients, receiving treatment for unrelated mental disorders. Here, a selected group of patients and a healthy control group were asked to record the times of migraine attacks they were subject to over a period of 2 weeks. Both groups were equipped with an actigraph watch. The study resulted in a relatively small number of subjects which reported their migraine attacks of which four were deemed to have produced labels consistent enough to be good for further analysis. The four subjects among them had experienced in total 28 migraine attacks. Several significant relationships were pointed out in the periods before, during and after the attacks. However, as a pilot study it based its findings on a small population. The sensor data from these

subjects, totalling over 25000 data points per person (using 1-minute epochs) was to become the basis for a large part of the material analysed in this work[2].

Attempts have been made to find relationships between miscellaneous physiological variables and migraine. For example,[73] in their migraine-related study, make an attempt to find connections to blood pressure measurements. A group of 62 normotensive migraine patients with and without aura take part with little success. In this case, no activity measurements are involved. Even though other sensor technologies are used, we find it interesting to observe the analysis methods used and to obtain potentially useful domain insights.

In another study, an ensemble of sensors, actually a *wireless body sensor network* (WBSN), comprised from four signals: heart rate (HR), electrodermal activity (EDA), skin temperature (TEMP) and peripheral capillary oxygen saturation (SpO2) was used in an attempt to predict migraine attacks, [74]. The results showed TPR (True Positive Rate) of just below 70% with PPV (Positive Predictive Value) of 100%, with a predictive horizon of up to 50 minutes prior to a migraine event. These findings support amongst other things the idea that ML models must be trained as per individual case, in accordance with the fact that behaviour of the autonomous system depends on the individual patient. Secondly, the study confirms the assumption that detectable variations precede the actual beginning of pain in the hemodynamic variables (i.e. features) of sensor measurements.

This supports the viability of our prediction assumption. Any similar relationships connecting early patterns and/or statistics in physiological variables to later symptoms are especially interesting to us in the context of feature extraction for the purpose of prediction of these symptoms. It is for this reason that a wide range of areas have been examined for our background research.

A study by Bruni, O. et.al. [75] was an attempt to discover and evaluate the relationships between sleep and migraine using objective actigraphy measurements over a considerable period of time. Previous studies were mainly based on parental ratings and subjective self-reporting. Quality of sleep in children with migraine was evaluated during the periods between and during attacks and patterns of sleep preceding, accompanying and following migraine attacks. Eighteen common migraine (without aura[3]) patients were compared with a control group of similar, healthy children. Actigraphy recordings of sleep parameters were taken over a period of two weeks with self-reporting diaries for event labelling and description of details of headaches. Over 57 attacks were recorded in total over the given period. The study went on to conclude that there was an observable reduction in measured nocturnal motor activity the night preceding a migraine attack, which can be seen to support the hypothesis of lower cortical activation. Decrease in cortical activation has in earlier migraine causality studies been

[2] We state the reason on several occasions: Due to the unexpected absence of adequate sensor data from bipolar studies (on time), a dataset from a migraine-study involving bipolar patients has been adapted for the experiments.

[3] Warnings of coming migraine: visual symptoms such as flashing lights, zigzag lines, blind spots in vision, also distortions, shapes. In some cases, tingling, pins-and-needles sensations in arms or leg.

accredited to typical premonitory symptoms (both subjective and neurophysiological), such as growing irritability, depression, withdrawal, tension, and tiredness.

3.1.3 Sensor data analysis

Traditionally, statistical methods and statistical modelling have been widely used in BP-related research. Even so, there is uncertainty in the underlying processes observed in time series data produced from such studies [76]. In a study of dynamics of mood regulation in patients with bipolar disorder the author suggests that nonlinear measures could be applicable in the study of mood disorders [77]. A different approach to study of actigraphy data, using statistical functional analysis is presented by [78] as applied "to assess the impact of apnoea-hypopnea index (apnoea) and body mass index (BMI) on circadian activity patterns measured using actigraphy". This method shows potential to improve the quality of analyses of circadian activity rhythms from actigraphy data extending its use domain from "general sleep assessment". Refined non-linear analysis methods are more applicable for biological time series, as its characterized by noise and non-stationarity [12]. Machine learning is a powerful alternative to traditional statistical modelling as it handles linearity and non-linearity equally well. We will apply both supervised classification and unsupervised outlier detection machine learning techniques in this work. In our classification experiments however, we will be concentrating on feature extraction from predefined segments of the data sequence.

An important distinction to make when using machine learning or indeed any method of accelerometer data analysis, is that of the environment where the data is or has been collected. Most studies (as observed by the author of this work) tend to use data collected in laboratory (controlled) setting, while it can be safely stated that the intended use of the resulting methods usually is targeted at *free-living* environments, i.e. where subjects are monitored under free-living conditions at home and work. The ecological validity of these methods can be thus improved [79]. In this study classifiers are trained to predict five activities: sitting, standing, bicycling, transport in a vehicle and walking/running. The results based off a controlled dataset and the free-living dataset were compared, demonstrating significantly better scores of the first over the latter with 89% vs. 71% accuracy score. The main conclusion from the study is that if the intentional target of usage is outside the laboratory-setting, then "it is important to train classifiers on free-living data."

Often, the environment setting cannot be easily controlled, but it is worth keeping in mind that any experiment results should be seen in context. A study Karam et.al.[11] suggests that the states of hypomania and depression in bipolar disorder are discernible from euthymic states by means of speech classification over a cellular phone. The classifiers were trained on both structured (carefully prepared clinical interactions) and unstructured cell phone recordings. Their findings were found to be more accurate with the structured (and labelled) data - the study being somewhat complicated by personal privacy issues concerning recordings of conversations

over cell phones. In summary, the system is able to detect hypomania through the unstructured collected in proximity of the structured interaction of a given day. The result scores (Area Under the Curve AUC) are in the mid-sixties, which may not seem impressive, but this is a very promising research in progress. Nonetheless this is a further step in support of the idea that underlying neurophysiological states can potentially be classified by identifying time series features from speech as predictor variables. The challenges encountered here have similarities to studies of actigraphy data: high variance, noise and individual differences, to name a few.

In a feasibility study of actigraph parameters relationship to relapses in bipolar disorder [5], the authors conclude that circadian rhythms inter-daily stability is a relatively useful indicator of prodromal symptoms in bipolar disorder transitions. This study spans over a period of 150 months in eight patients, with 17 transitions (relapses) recorded. It is mainly interesting because of the scale (over time) of the measurements collection. As for the accuracy, the reported sensitivity/specificity scores were 65% and 68% respectively, but only 3 patients produced readings of over 12 months and even in these missing data was a problem. Even though this study can hardly be considered significant, it nevertheless adds some support to the idea of prediction.

3.1.4 Classification

The traditional use of the actigraph primarily for determination of sleep-wake cycles and quality of sleep has been challenged on occasions. Recognition and measurements of activities outside of the sleep domain have gained favour. One such study has used actigraphy in combination with state of the art machine learning algorithms to analyse activity signals in an attempt to classify the most important activities of daily life (ADL). Human activities such as eating, hygiene, cooking, watching a movie and sleeping were assigned separate classes [80]. A major effort was put into the feature extraction process, including curve fitting for each activity class, smoothing, statistical features and time of day information. The sequence of events was also taken into account, improving the accuracy of the predictions. Finally, a comparative study of the four machine learning algorithms is made based on the same set of extracted features [80]. The results were very encouraging with the winner algorithms (“Logitboost”) giving accuracy results of just above 90%. An important lesson learned from this study is the utilization of timestamp information in the feature extraction process, something we assume could be significant in detecting patterns over time.

In a classification task one of the main factors towards obtaining accurate and useful results is the feature extraction and selection process [43]. We have found several studies supporting the idea that this area is critical for improving classification algorithm performance. The improvement in classification accuracy of transport mode exclusively through the use of smartphone accelerometers has been attempted in a study by Hemminki et.al.[81]. Although

applied to a different domain, the technicalities of advanced feature extraction presented could prove relevant in further work on improving performance in classification of accelerometer data.

In "*Classification and Feature Analysis of Actigraphy Signals*" [43], they compared a selection of 63 different features that were found to be used in analysis of actigraphy signals. Then, two feature selection techniques were applied to rank and compare their effectiveness and subsequently used with two distance-based clustering algorithm classifiers to classify activities of daily life (ADL) signals. The study concluded with accuracy ratings of around 95% for a subset of just 5 features or less, at the same time stating that the statistical features (such as mean, n^{th}-percentile, quartiles) were repeatedly among the best performers. Considering the low cost of implementing simple statistical features and their reported effectiveness makes them doubly attractive for use in classification experiments.

In working with the migraine dataset, we were soon faced with another challenging problem, namely that of class imbalance in context of classification algorithms. Class imbalance entails that one or more classes in a dataset are strongly overrepresented in relation to the others. As an example, in critical applications, such as in medical cancer diagnosis misclassifying cancerous cells as non-cancerous may lead to very serious consequences. Most algorithms will currently lean to the advantage of the majority class, with some SVMs performing slightly better than most. [25, 62]. There is however evidence that through specific weighting and boosting techniques, there is room for some improvement. This is demonstrated in a study by Chen et.al. [32], where basic Random Forest classifier is tested and evaluated with weighting and balance (based on down sampling of majority class) modifications. The results obtained show a clear advantage for the modified RF implementations and the authors conclude that both Weighted Random Forest and Balanced Random Forest have performance superior to most of the existing techniques that we studied.

In general the normally proposed solutions to the class imbalance problem seem to converge on four main areas: the preprocessing stage over-and-under sampling techniques, introduction and tweaking of weights in the algorithm, tweaking the decision boundaries using performance metrics and use of ensemble methods. [82]. According to [62], boosting techniques are powerful ensemble learning algorithms that improve the performance of weak classifiers. Furthermore, data preprocessing provides a better solution than other methods as it allows for the addition of new information and/or deleting of redundant information. In conclusion, the study suggests that applying multiple techniques gives better overall results in class imbalance problem.

3.1.5 Anomaly Detection

As mentioned in the introduction, Hierarchical Temporal Memory (HTM) and in particular, its anomaly detection potential will be given some room in our experiments. The assumption is

that adverse events (such as a bipolar phases or migraine attacks) influence the physiological state, and therefore also activity patterns. Abnormal patterns can then potentially be detected as outliers, i.e. anomalies, given enough historical "normal" data.

Statistical methods, such as relative entropy were used for comparison of historic distributions against current, adequately windowed distribution with some success in anomaly detection task for a big data centre [83]. The results were compared to more traditional threshold methods in measurement of metrics such as CPU and memory utilization. Even though the underlying signal complexity is of a completely different nature, we find many parallels between the approach used here and in measurements from biological sources.

Anomaly detection is an alternative (in the precise sense of the word!) method to supervised human behavioural change detection techniques. In a related study, "a reliable method to detect disorders and diseases in healthcare applications" [84] is proposed, employing an automatic "abnormality detection" method based on physical activity measurements. This method in a similar fashion to HTM is based on changes in physical activity relative to the historical data [59]. No assumptions about the underlying data distribution are made. Behavioural changes are evaluated in real time with high precision and recall scores (100% and 92%, respectively), notably on a small population (N = 3), but ranging over a time period of 3 months per subject. It is worth noting that abnormal behaviour can be considered a subjective notion and dependent on some agreed threshold for what is considered (by each individual) to be normal. The fact that this study builds on statistical past to classify the current from activity data was of main interest here.

3.1.6 Analysis tools

The analysis of time series sensor data poses some challenges, beginning at preprocessing stage through the training/analysis and ending in evaluation and presentation of results. The preprocessing stage is often considered the most time consuming and critical as the rest of the process is dependent on adequately prepared input [42]. Several tools and programming languages were considered for the task. Some research into the general domain of Data Science has revealed a whole range of possible choices. The three most popular, complete software packages are RapidMiner, R (with RStudio) and Weka. R is a programming language traditionally used for mathematical, scientific & statistical computing and visualization with a massive number of libraries. RapidMiner and Weka are more user-friendly, GUI-based with varying choice of features and drawbacks. These three tools are compared in [85]. More flexible programming tools have recently gained popularity in the data science community: Python with several machine learning and statistics libraries, such as Scikit-Learn, Keras and quite recently, Tensorflow by Google, the latter two predominantly used for working with deep learning neural networks. In the case of Numenta's HTM, Python is required to use the official

API. R and Python have the advantage of flexibility and crucially, massive support of libraries and support forums so these were mostly used for creating experiments.

From this section, we gained knowledge of methods used in previous studies involving actigraphy as well as other sensor types. We found ideas for implementation of potentially useful features relating to migraine and bipolar disorder. Furthermore, we found studies with some success in early detection of symptoms, specifically of migraine. This supports our assumption that there exist patterns giving early warnings, though not necessarily through exclusive use of the actigraph. We saw non-linear machine learning methods suggested as having potential in analysis of actigraph data, gained specific information on most effective statistical features and methods for tackling imbalanced data. Finally, we have touched on anomaly detection methods and reviewed software tools that are currently used for machine learning data analysis.

3.2 Machine Learning

Machine learning in the most general of meanings is a set of tools and methods with the aim of modelling and understanding data. Machine learning algorithms can learn how to perform important tasks by generalizing from examples. This is often feasible where manual programming is not. As more data becomes available, more ambitious problems can be tackled [15]. Machine learning algorithms have the ability to extract information and infer structure from collections of measurements, data streams such as sound/video signal and human language. In recent years, machine learning has increasingly shown its usefulness in a wide range of areas [16]. Machine learning is the force behind the Big Data paradigm [17]. The terms, "Data Mining" and more recently, "Data Science" are increasingly used to describe the process of applying machine learning methods for all kinds of data analysis. Such interrelation of ML and statistics lead to the overlaps of frequent terms as shown in Table 3-1Table 3-1: An unofficial summary of interrelated terms used in data science community and literature. (based on source: https://www.analyticsvidhya.com):, below.

Machine Learning	Statistics
Network, graphs, classifier	Model
Weights	Parameters
Learning	Fitting
Generalization	Test set performance
Supervised learning	Regression / classification
Unsupervised learning	Density estimation / clustering

Table 3-1: An unofficial summary of interrelated terms used in data science community and literature. (based on source: https://www.analyticsvidhya.com):

3.2.1 Classification Algorithms

We begin with a simple description of supervised learning or classification. In data science, a *feature* is a term used to describe an attribute of a member of a set. Such a set can consist of one or more structurally similar subsets, usually referred to as "classes". The class membership of each individual member in the set is decided by the value of the attribute. Naturally, set members may potentially carry multiple attributes. In *supervised learning*, a vector of attribute values in a training set is paired with a target class label[4], indicating its class membership. This complete vector of attributes is known as an instance or training example. Training a classifier involves feeding training examples to the algorithm so that it can create rules to assign membership from the individual attribute values. The algorithm attempts to find a fitting function from input to the respective targets, thereby creating decision boundaries between the classes. Provided a new, unseen example, the trained classifier will infer from the model it has built, what the target label for this example should be. The label output obtained from the classifier is also known as a prediction.

These characteristics are common to all supervised learning (classification) algorithms. Below, we present short descriptions of the individual algorithms that will be used.

3.2.1.1 Gaussian Naïve Bayes

Naive Bayes (as in Bayes theorem) is a simple probabilistic classifier based on the 'naïve' assumption that the values of the individual features are independent of the other features on a pair-by-pair basis. This assumption is most often violated in real life problems, but it has been shown that even in these situations the algorithm performs surprisingly well [18]. Simple as it is, Naive Bayes has in the past managed to outperform even highly sophisticated classification algorithms. It belongs to the parametric machine learning methods [19]. NB is simple, models are relatively easy to build, require less training data, are fast, efficient and so useful for very large data sets. Additionally, the algorithm produces results as probability distributions [20]. Both, binary and multiclass classification tasks are handled well by this classification algorithm. "*GaussianNB*" as implemented in the Scikit Learn-library, implements the Gaussian Naive Bayes algorithm for classification where the likelihood of the features is assumed to follow a Gaussian (i.e. normal) distribution. "GaussianNB" requires only that the mean and standard deviation estimates are extracted from the data [20]. This implementation requires virtually no hyperparameters and is one of the simplest used in the experiments.

[4] The short version «target label» is used throughout this document.

3.2.1.2 Logistic Regression

Logistic Regression (LR), though in some ways similar to *linear regression*, is in fact a binary classification method and not a regression model. As a parametric generalized linear model, it is simple and has good performance. The binary response variable is related to any number of explanatory variables which when combined, form the basis of probability calculations that predict the target class, i.e. in this case the categorical response variable. Also, known as maximum-entropy classification method or logit regression, named after the decision function used in the classifier. The logit function is the natural log of the probabilities that the target variable takes one of the two values [21]. Multinomial logistic regression, also called SoftMax regression is a form generalization of LR which deals with cases of multiple target classes. A well-behaved classification algorithm, LR is generally considered a good baseline choice when given features are linear and the problem is potentially linearly separable.

3.2.1.3 Support Vector Machines

Support Vector Machines (SVM) is a set of supervised learning methods used mainly for classification, regression and outlier detection. SVM are non-probabilistic, linear methods that find the optimal decision boundary using just a few bounding members or training points of each class. It is from these few, critical points, called "support vectors", that the algorithm gets its name. SVM is considered to be accurate, guaranteed to reach the global optimum, generalize well with adequately chosen parameters. The main advantage of SVMs comes from the fact that through using the so-called kernel trick they are perfectly capable of being effective non-linear classifiers [22]. A kernel is basically a function which quantifies the similarity of two observations [23]. The method effectively creates produce nonlinear boundaries by constructing a linear boundary in this potentially infinite-sized transformed version of the feature space [24].

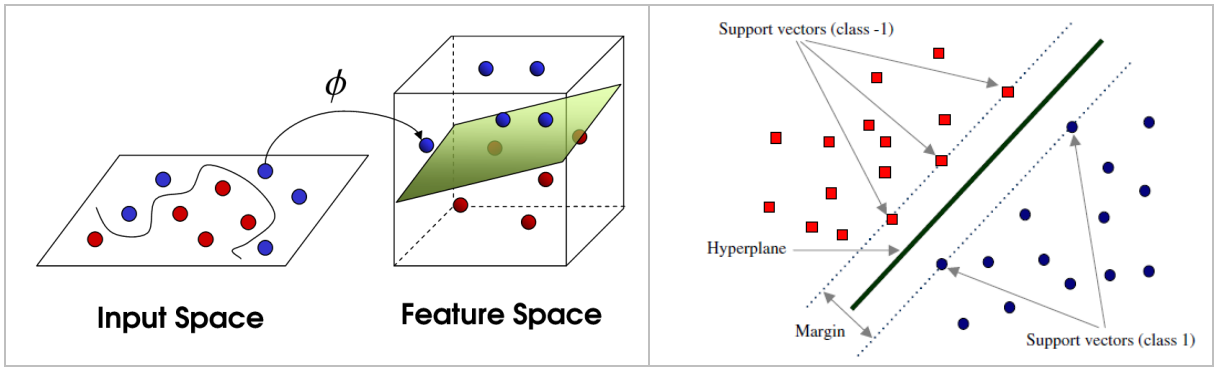

Fig. 3-1: Left: Simple diagram showing how transformation of feature space effects in non-linear classifier. Right: Representation of the margin boundary. (image source: https://amitranga.files.wordpress.com)

The boundaries created this way are the hyperplanes (lines, in the simplest of cases) that maximize the distance between the two classes, also called margins (see Fig. 3-1). SVMs are effective working with high dimensional spaces, they are memory efficient, very versatile because of the choice of various kernel functions that can be utilized for the decision function [22]. Among the disadvantages of SVMs is that number of features exceeding the number of samples might lead to poor results and the inability to provide probability estimates. Most importantly, SVM classifiers have been reported to underperform on problems involving heavily imbalanced data [25].

3.2.1.4 KNN

Nearest Neighbour is a simple, flexible, non-linear classification algorithm and a typical example of an instance based learning method. Although simple, KNN (prefix K relates to the one major parameter required, described below) has been successfully applied in a large number of classification problems, including EKG patterns, handwritten digits and satellite imagery [24]. Instance based learning methods are sometimes called "lazy" because processing of data is delayed until new instance arrives due for classification. The K-Nearest Neighbour algorithm is based on an assumption that data instances can be represented as points in n-dimensional space and the distances between points can be measured [26]. For instance-based learning in general, no explicit model building is done on training data; the process of learning consists of storage of the training instances until the time when a new instance is due to be classified. At this point the stored instances are searched for potential matches (nearest neighbours) to the new one, utilizing a distance metric such as Euclidean distance as a similarity measure [27]. The k-parameter (also called the radius) in the k-nearest neighbour represents a distance threshold as pertaining to the number of nearest neighbours that will be considered during a search. Providing increasingly higher values for k, increases the bias (reducing the variance) of the decision as it considers more information but at the same time increases the processing time while the model complexity, paradoxically, decreases.

3.2.1.5 Multilayer Perceptron

Multilayer Perceptron (MLP) is a feedforward[5] *artificial neural network* (ANN) variant that is perhaps the most used today. While a single perceptron is limited in its scope as it can only learn linearly separable functions, a network of perceptrons with suitably adapted activation function can become a very effective non-linear classifier. The step function of the traditional perceptrons is non-differentiable ("not smooth enough for optimization"), so as an alternative the sigmoid or logistic function has been introduced [24]. A multilayer perceptron model, as

[5] A feedforward neural network in which connections between the units do not form a cycle.

shown in Fig. 3-2, is composed of at least one hidden layer of neurons (as the name implies), generally increasing the predictive power (and complexity) with each added layer.

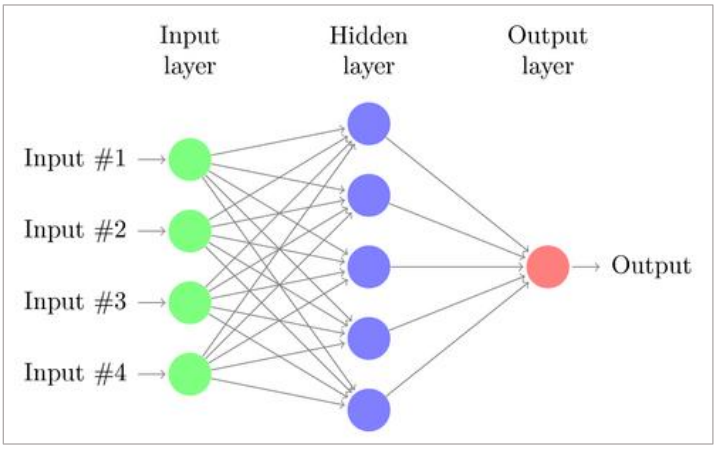

Fig. 3-2 The simplest MLP network with 1 fully-connected hidden layer. (image source: http://www.texample.net)

In MLP the *backpropagation*[6] algorithm with gradient descent is used to train the weights of the fully connected network [26]. The hierarchical, multi-layered structure of a neural network is the source of its ability to create mappings of complex, non-linear problems. Another advantage lies in the fact that features are derived directly from the training data that best represent the relation of the input vector to the target variable. A powerful general approach for classification (and regression) problems, neural networks have been shown to compete well with the best learning methods [24]. They are relatively robust to errors in the training data, because they are not learning exact rules, but minimize a cost function instead. Training times are generally much higher than for most other machine learning models, but on the other hand, an already trained classifier has constant *O(1)* execution time.

3.2.1.6 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simplification of the popular *gradient descent* optimization algorithm [28], maybe the most frequent way to optimize neural networks. Fundamentally, dradient descent is an algorithm that minimizes a loss function. A gradient is computed on a loss function by using differentiation, specifically calculation of partial derivatives of the function parameters. Basic gradient descent is known to get stuck on local minima in searching for the optimal solution. There are several variants, mainly differing on the amount of data required to compute the gradients. The trade-off is between accuracy and the time it takes to perform parameter updates. Stochastic gradient descent is an improvement of the original algorithm, that is both faster on large datasets and greatly enhances its effectiveness in finding the global cost minimum [29].

[6] A common method of training neural networks used in conjunction with some optimization method (such as gradient descent).

3.2.1.7 Ensemble Methods

Ensemble methods is a general name for multiple models or combinations of models combined to produce predictions that have the goal of improving performance and generalizability over a single model [30]. There are two main types of ensemble methods: boosting and averaging. In the first category, we find *Gradient Tree Boosting* and *AdaBoost*. Here the basic idea is to create a number of estimators in a sequential fashion, with the aim of improving on the predecessor. This way, several weak classifiers can produce a powerful single ensemble. The second category creates multiple estimators in an independent fashion with the goal to combine the resulting predictions and averaging them to obtain the final one. This has the effect of reducing the variance and increases overall robustness. Forests of randomized decision trees (see section 3.2.1.8), majority voting and bagging methods belong to the second category.

3.2.1.7.1 Voting Classifier

Combining several unrelated machine learning classifiers to get a single, optionally weighted result is the main idea behind the *Voting classifier*. This follows the general idea of bagging ensemble methods, but is not constrained to any single type of estimator. The individual estimators' predictions can be weighted in order to balance out the final prediction which in the case of a majority vote will be the one representing the majority of obtained results.

3.2.1.7.2 AdaBoost

One of the first proposed practical boosting ensembles and still among the best and most used and discussed is *AdaBoost* [31]. A critically important base assumption of AdaBoost is that the collection of weak classifiers produces varying above average (i.e. not random) predictions in each iteration. These weak classifiers are assigned weights, which are adaptively adjusted at the end of each iteration. The adjustment is relative to the target labels, effectively decreasing the weights for correct classifications and increasing for incorrect ones. The measurement of an algorithms confidence in predictions is called the margin and is in this case based on the majority vote model implied by the weights adjustment. AdaBoost ability to enlarge the margin and enhance its generalization capability is one part of its success. Several studies utilizing decision trees, neural networks or support vector machines as components report good overall generalization performance. One of the most attractive advantages of AdaBoost is its resistance to overfitting through implicit regularization, although it certainly is still possible to overfit [31]. AdaBoost can be used for both classification and regression tasks.

3.2.1.8 Decision Trees & Forests

Decision tree learning methods are one of the most common non-linear classification methods. Their popularity can partly be accredited to their intuitive representation of the decision-making process. Each internal node in a decision tree is labelled with an input feature or an attribute and represents a test resulting in a branch based on a certain threshold value for this particular attribute. Both the actual branch attribute as well as the critical split value are determined using an optimization procedure, though this may differ between various implementation types. The individual branches represent the outcome and lead to child nodes with subsequent tests, or in the case of a leaf node, a target class label. The hierarchical structure implies that the root node and the root nodes of each subtree will contain the most significant attribute test based on information gain (usually: entropy) calculation. While single trees are useful for demonstration purposes, it is ensembles of trees i.e. tree forests, that that are currently considered more effective [32]. Forests of trees have many advantages over single trees (in line with other ensemble methods described in pervious section), most importantly, lower error margin and better generalization. As with other non-parametric machine learning methods, decision trees in general require more training data and are more prone to overfitting than their parametric counterparts.

3.2.1.8.1 Random Forest

Developed by Leo Breiman in 2001, *Random Forest* is an ensemble of unpruned trees used for both classification and regression. The idea is to build as forest of decision trees or weak classifiers from bootstrap samples of the data. This method involves random feature selection for the purpose of building the individual trees and the final classification result is calculated from aggregating, (or averaging, in case of regression problems) over the members. Random forest has shown massively improved performance over traditional single tree classifiers [32, 33] such as C4.5 and J48, i.e. single decision trees. Because of the random, independent sampling used for building the individual trees, the generalization of forests decreases as the number of trees increases.[7]

3.2.1.8.2 Extremely Randomized Trees

In *Extremely Randomized Trees* (ET) algorithm, as in Random Forest, a random subset of attributes is used but the decision for splitting a node is based on purely random thresholds, not necessarily the one with the best discriminative power. Additionally, ET uses the complete learning sample as opposed to the default bootstrapping method employed in other tree-based

[7] T*he Scikit-learn library implementation of Random Forest combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class* [20]*.*

ensemble methods like RF. The bias-variance trade-off is shifted here towards reduced variance at the cost of higher bias error. The advantages of extremely randomized trees algorithm are great accuracy and attractive performance speed. ET has already been noted for its effectiveness in problems of high dimensionality, such as time-series and image classification [34], the first mentioned quite relevant in this work.

3.2.2 Hierarchical Temporal Memory

Unsupervised learning entails that the machine learning task is supplied with a set of instances, but the target labels are non-existent, or otherwise not given as reference. Traditionally this is the domain of clustering algorithms which form groups (or clusters) from the unstructured data, effectively "discovering" target classes. *Outlier detection* is another, more specific form of unsupervised learning, as the name states focused on finding the extremities or outliers in the data. *Anomaly detection* is an alternative term for outlier detection. Extremely rare events, patterns and values are general examples of outliers, but the specific meaning is domain dependent, e.g. credit card fraud detection from millions of transactions. Generally, when the goal is to discover few positives among a massive number of negatives, one should consider this category of algorithms. The *Hierarchical Temporal Memory* (HTM) algorithm belongs in this group.

About six months before starting on this project, we came across information on a project by a US company, *Numenta*. Numenta is a company doing research on *Artificial Intelligence* (AI). Their approach is based on building a model that attempts to simulate the way the neocortex of the human brain as closely as possible. One of the results of Numenta's research is the Hierarchical Temporal Memory algorithm, which was claimed to be among the best performing in context of temporal sequence and pattern detection [35]. The *Hierarchical Temporal Memory* (HTM) earlier known as *Cortical Learning Algorithm* (CLA), is a machine learning algorithm taking a very different approach to the "traditional" methods of machine learning and artificial intelligence.

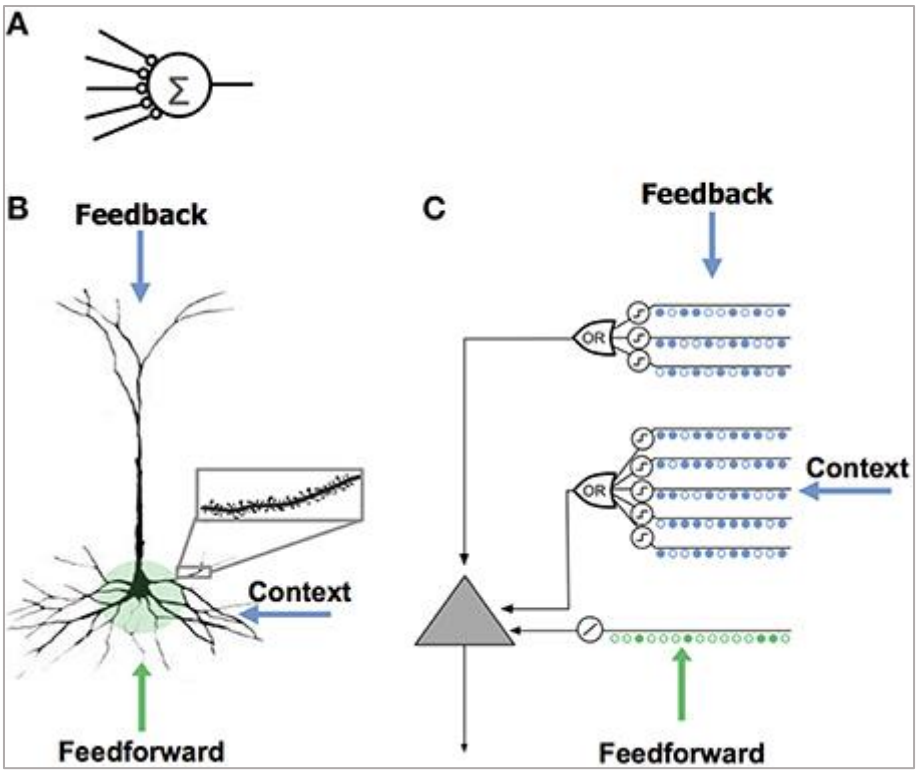

Fig. 3-3 Comparison of neuron models in ANN vs HTM. (A) The neuron model used in most artificial neural networks has few synapses and no dendrites. (B) A biological neocortical pyramidal neuron has dendrites (inset zoom) covered with thousands of synapses. The co-activation of a subset of synapses will cause a form of an electrical signal called an NMDA spike [36] *and depolarization at the soma (highlighted spherical part). There are three sources of input to the cell. The feedforward inputs (green) which form synapses proximal to the soma, directly lead to action potentials. (C) An HTM model neuron models dendrites and NMDA spikes with an array of coincident detectors each with a set of synapses (only a few of each are shown)* [37].

HTM is an attempt to use the human brain as the de-facto model for the way it recognizes, detects and predicts patterns. This biological model implies deeper understanding and emulating parts of the human cerebral cortex: the part of the brain known to be responsible for functioning of memory, attention, perception, awareness, thought, language, and consciousness. However, one could argue that neural networks already adopted this idea long before. This is only partly true as the idea of the perceptron was indeed based on neurons in the brain, but it was very simplified, omitting a lot of details (see Fig. 3-3) not well understood at the time. HTM builds models much more closely based on the structures found in the cortex [37].

Although it can be applied to both classification and regression type of problems, we will focus on its ability for outlier detection or otherwise known as anomaly detection. An anomaly detection algorithm is tasked with the detection of abnormal events, sensor values, operational states or any combinations of these. Automatic anomaly detection is preferable in areas where it is not feasible for a human expert to explicitly write rules for alarm initiation or where rules can be derived from the (normal) data distributions. The algorithm can effectively learn and build a model of normality, given enough historical data. When encountering new instances, they can be classified (as abnormal) based on this information.

3.2.2.1 Sparse Dynamic Representation (SDR)

HTM uses a specific underlying data-structure called Sparse dynamic representations (SDR). SDRs differ from standard binary representations such as ASCII, as the meaning is encoded directly into the representation. SDRs are based on large arrays of bits, where the majority are zeroes, the rest being ones. "Sparsity" is the measure of the ratio of on bits to off bits and is around 2% in case of HTM. Every bit is assigned with some meaning, an overlapping of two SDRs with on bits means they are semantically similar [38]. SDR has several important characteristics including their high capacity, low mismatch probability, very low probability for false positives, high tolerance to errors with noisy data, union operations of SDRs, robustness of unions in the presence of noise [39]. This is exactly the kind of characteristics required for successful detection of anomalies in a sensor data stream.

3.2.2.2 Encoding

To be useful for processing, the data stream needs to be encoded into a SDR which the HTM algorithm internally requires. If the data can be converted into an SDR then it can be used by any application based on HTM [38]. Depending on the data and its intended use the encoder can be very simple or very complex. Either way the output SDR have underlying semantic meaning, for example: for a scalar encoder ranged from 0 to 100, where the values are of importance, the digit 5 and 6 have much more in common than 5 and 7, and would have nothing at all in common with 50. Again, this heavily depends on what it is the data actually represents and the meaning we wish to assign to it.

The basic encoding scheme for encoding a range of numerical values (*Scalar encoder*) has at least four parameters: min value, max value, number of buckets, total active bits. The number of buckets should be chosen depending on the expected inherent noise for this metric. Prediction quality (accuracy) depends on this parameter: a noisy signal suggests a smaller number of buckets, giving the HTM more stable input at the cost of less precise predictions. A very clean signal, given a large number of buckets would cause HTM to be able to make more precise predictions [38].

Appropriate encoding technique is of great importance for applications as the number of classification variables increases. Using date encoding makes it possible for us to supply context to patterns over time (temporal patterns), as we demonstrate in section 5.1. In this way, the date encoder assist in automatic discrimination of time periods, days, weeks, weekends, months, etc and correlate these with potential periodic patterns.

The Delta encoder encodes the change in value between two consecutive data points rather than the individual values of the data points. This is helpful in recognizing patterns in data independent of individual magnitude, for example relative temperature change patterns or finding patterns in rising or retreating stock market indices. Other types of encoders include Categorical, Geolocation and Random Distributed encoders. Depending on the problem domain, it might be advantageous to create a new or modify an existing encoder.

3.2.2.3 Spatial Pooling & Temporal Pooling

Spatial pooler groups together similar inputs, i.e. SDRs with overlapping bits [40]. Two inputs are semantically similar if they have a specified number of overlapping bits in their representation. Attributes like threshold can be adjusted accordingly although, there has been done extensive testing and for most purposes the pre-set values give satisfactory performance.

Temporal Pooler (TP) learns sequences or transitions between SDRs [40] in a sequential manner, i.e. over a period of timesteps. The temporal pooler output predicts unions of SDRs. What this implies can be explained with a simple character sequence example: after being trained on a sequence "ABACAD", the TP when it encounters an "A" at the next step in its input, it would predict the union of "BCD" to be a possible next value. This would set them corresponding bits in what is called the predictive state. TP is a component of high-order memory, as prediction of following SDRs is based on the history of SDRs already encountered before. To illustrate with a modification to the previous example: with historic inputs of "ABCD" and "XBCY", when the sequence "ABC" is encountered, then a "D" will be predicted instead of union of "Y" and "D".

Finally, the outputs from the temporal pooler are processed by the CLA-classifier (more recently, revamped as the SDR-classifier). The classifiers task is creating a single prediction from the current encoded SDR state of the TP. The classifier operates on tables of probabilities of the occurrence of each SDR, basically choosing the one with the highest frequency for a given context, at the same time decoding the output to match the original input type.

3.2.3 Feature engineering

A feature is a property, sometimes called an attribute of individual instances in a training set. Wikipedia gives one definition as: "*In machine learning and pattern recognition, a feature is an individual measurable property of a phenomenon being observed…".* [41]

The explanatory variables in a simple linear regression problem can be considered as features. While in many cases, features can be derived directly from the explanatory variables i.e. attributes of a structured dataset, in other cases that data might be raw sensor data such as that obtained from an actigraph. In this case features have to explicitly extracted from a time series and that is part of a process of feature engineering. Feature engineering (also called feature construction), while not a formally defined term is nonetheless a critically important part of the preprocessing set of operations involving extraction, creation and selection of features for machine learning applications [42]. Optimally a set of features should be maximally independent, informative and discriminative in order to improve the performance of applied classification algorithms.

3.2.3.1 Feature Extraction

There is vast literature on various feature extraction and feature selection methods, the subject is vast and beyond the scope of this work to describe in full detail. We will however include some important general takeaways. In this work, we divide types of features into three somewhat overlapping subcategories:

Domain-specific features, where relevant knowledge about the problem task is used to identify potential points of interest in the data. A slightly oversimplified example of a domain feature would be the classification task of dogs and cats, where the critical piece of domain information made available to us would be the fact that cat eye pupils contract into a vertical line when exposed to strong light. In this case, this is the top discriminative feature that probably would give a 100% accurate classification result with the simplest of classifiers. On the other hand, information like weight, height and life span would be adequate but not specific to the domain.

Statistical features require no special knowledge of the domain, but still provide useful characteristics in the form of quantitative data based on properties such as mean, standard deviation, median, maximum and minimum [43]. Depending on the aim of the analysis a time series can be further decomposed and further information extracted through transformation methods like Fast Fourier Transforms (FFT) w.r.t. the frequency domain [44]. Statistical variables can be readily calculated making them popular and computationally cheap to use.

A different but fully dependent approach involves adding polynomial features. Polynomial features represent all possible interaction between 2 existing features and the square of the

original features. Polynomial features [45] improve linear methods accuracy on non-linearly separable problems (Fig. 3-4). In practice, we derive these additional features indirectly, by using the statistical feature values calculated on the time series segments.

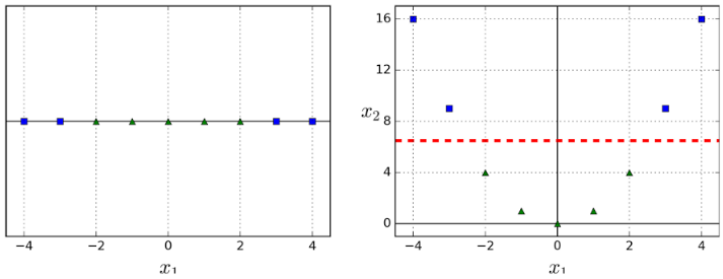

Fig. 3-4: Example showing how adding (polynomial) features may make data linearly separable. Source: [45]

Visually derived features are usually based on early exploratory analysis, where some visibly "uncommon[8]" patterns in the data have been spotted. Depending on the problem task such observations could be potentially useful for determination of additional features (such as the one in section 4.3.3.3) or maybe subject to filtering out.

Feature extraction in practice has some creative aspects to it. It provides freedom to brainstorm new ways of obtaining potentially useful features, but in general it might involve a lot of manual work, testing and validation as well as use of statistical tools and automating tasks. Additionally, combining existing and/or derived features is a viable method to further cover aspects that might not be obvious at first inspection. For example, the addition of polynomial combinations and/or logical conjunctions of existing features can improve the ability of a linear classifier to model nonlinear problems [46].

3.2.3.2 Feature Selection

It has been shown that while increasing the number of features in general has a positive effect on classification accuracy, a high number of features can also have a detrimental effect on the accuracy performance of a classifier. The key idea is feature relevance. Somewhat contrary to the idea of creating new features, feature selection is concerned with the aim of reducing the number of features made available to the classification algorithm. The motivation behind this comes from the fact that increasing the size of feature vectors, proportionally increases the amount of processing to be executed by the ML algorithms subsequently applied to the data. There are other important considerations to keep in mind as the number of features grows, such

[8] There might be a certain overlap with the domain feature category here to be able to make the objective decision of what in fact is unusual pattern for a given dataset.

as the ability to measure their individual importance or discriminating power in the classification task [47].

Unlike other dimensionality reduction techniques such as *Principal Component Analysis* (PCA) or compression techniques from information theory, feature selection does not change the underlying structure of the data. As the name implies, features are selected out based on some measurement of importance depending on the specific feature selection algorithm used. Filter methods, as they are more generally called are based on correlation and mutual information calculations. Main advantage of filtering techniques is scalability, low complexity, performance speed and the fact that they are independent of the classification algorithm. On the more critical side, filter methods tend to ignore feature dependencies. The *Correlation-based Feature Selection* (CFS) algorithm is an example of an improved filter, with respect to speed that also takes into account the intercorrelation among individual features [48]. Alternatively, unwanted features can be eliminated by some iterative or recursive evaluation process, such as can be found in *Recursive Feature Elimination* (RFE). This second category also called the "wrapper" methods use greedy and heuristic search methods to produce the final selection relative to the classification model. These methods are however considered highly expensive computationally [47].

Finally, we have the embedded methods of feature selection, which are fundamentally selecting the set of features implicitly when building a model. This makes the selection specific to the chosen learning algorithm but is less computationally expensive than the wrapper method. Decision tree based algorithms and weighted Naïve Bayes are good examples [47].

Feature selection methods applied in our experiments are based on implementations from the feature selection libraries of Scikit-learn [20].

3.2.4 Evaluation & Metrics

Model evaluation is a crucial part of model development. For any given prediction model the goal is to build a model that predicts the target value for unseen data instances with the highest possible accuracy. We need a way of evaluating model performance, typically by quantifying it with some measure of model error. This same measure should be used to train the model to obtain good performance. One of the great pitfalls in model creation is to base the evaluation of model performance on the very same data that it has been trained on. [49]. Using incorrect error measures and evaluation methods can lead to generation and selection of overoptimistic and overfitted models, i.e. incorrect models.
In general, the two main types of error are Bias error: overly generalized model, underfits the data and Variance error: overfits the data, the measure of sq. deviation over the mean. There is a trade-off between them and a good model minimizes both [50]. Specifically, we want a model that is valid, it measures what it was intended to measure and generalizes well to unseen data.

3.2.4.1 Model Validation

There are two main accepted methods of evaluation of machine learning models: the hold-out method and cross-validation. In both cases a separate test set of unseen data is used in the process of evaluating model performance.

While training a model, whichever metric is chosen the objective is to minimize the training error (or maximize some accuracy score). Using this result for final evaluation, i.e. training and testing on the same dataset is easy, requires few resources, but results in overfitting.

The *Hold-Out method*, also called a train/test-split, requires that a part of the original data is held-out from the training process (a test set) and that the final evaluation score is only calculated from the model performance on that test set. This method is also relatively fast and simple, and it ensures that the model has tested on unseen data. The disadvantage here is that a part of the data is removed from the training set to the model. Furthermore, there is risk of relatively high variance of the predictions.

Fig. 3-5 10-fold cross validation. The designated training set is further divided up into K folds (K=10), each of these will now function as a hold-out test set in K iterations. Finally, the scores obtained from the model on individual iterations are summed and averaged into the final score. (image source: https://sebastianraschka.com)

Cross validation [24][26] is a further extension of the hold-out method. Cross validation is widely accepted as the state-of-the art method for ensuring model reliability and improved generalisation ability. In *k*-fold cross validation (Fig. 3-5), *k* is the number of partitions that the training set will be split into. The same *k* is also the number of iterations that the classifier will execute, each resulting in an evaluation score. The *k* scores are then averaged to obtain the final (training) score. For each iteration one of the *(k-1)/k* of the data is used for training the classifier, while the remaining partition of size *1/k* is used for validation of the model from that iteration. With each iteration, the next, yet unused partition is set aside for validation test, and a new model is trained from the remaining partitions as described above. The final score is then averaged over the number of iterations, *k*. The advantages of *k*-cross validation are reduced variance because of the averaging effect, but the process is slow having high resource requirement, as the classifier has to be trained and multiple (*k*) times over the total size of the training data.

3.2.4.2 Metrics

A good model measures what it was intended to measure and generalizes well to unseen data. In this section, we will introduce the set evaluation metrics that will be used in the experiments. The main motivation for choosing relatively many different metrics is to encourage comparative experiments and provide an extended spectrum of potentially complementary information. According to previous analysis by [51] it can be shown that most of the metrics commonly used in machine learning for evaluating classifiers, fundamentally measure different things, this being especially true for multiclass and imbalanced class problems.

3.2.4.2.1 Confusion Matrix

A *confusion matrix* presents a complete and very intuitive overview of the classifiers performance. The matrix is of dimension C * C, where C is the number of target classes in the problem.

		predicted condition		
	total population	prediction positive	prediction negative	Prevalence $= \frac{\Sigma \text{ condition positive}}{\Sigma \text{ total population}}$
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)	True Positive Rate (TPR), Sensitivity, Recall, Probability of Detection $= \frac{\Sigma \text{ TP}}{\Sigma \text{ condition positive}}$
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**	True Negative Rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ TN}}{\Sigma \text{ condition negative}}$
	Accuracy $= \frac{\Sigma \text{ TP} + \Sigma \text{ TN}}{\Sigma \text{ total population}}$	Positive Predictive Value (PPV), Precision $= \frac{\Sigma \text{ TP}}{\Sigma \text{ prediction positive}}$	False Omission Rate (FOR) $= \frac{\Sigma \text{ FN}}{\Sigma \text{ prediction negative}}$	Positive Likelihood Ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$
		False Discovery Rate (FDR) $= \frac{\Sigma \text{ FP}}{\Sigma \text{ prediction positive}}$	Negative Predictive Value (NPV) $= \frac{\Sigma \text{ TN}}{\Sigma \text{ prediction negative}}$	Negative Likelihood Ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$

Fig. 3-6: An illustrative depiction of the (binary) confusion matrix and a selection of the measures that may be derived directly from it.

The ground truth is matched against the predictions from the model, effectively showing information on how accurate the predictions are for each class and the distribution of misclassified instances for each class. The confusion matrix is the basis for most evaluation metrics that are frequently used in machine learning, one exception is the probability distributions that can be obtained from most algorithms, something we will come back to in section on ROC curves (section 3.2.4.2.5). In a binary (two class) example (Fig. 3-6), four individual basic counts are obtained from the basis of the matrix: True Positives, False Positives, True Negatives and False Negatives (TP, FP, TN, FN, respectively). From these we can derive most of the metrics described below.

3.2.4.2.2 Accuracy

The most frequently used and cited evaluation metric, *classification accuracy* is the percentage of correctly classified examples out of the total number of examples. *Accuracy* (ACC) is a perfectly good metric if the class distribution is well balanced, i.e. in the case of a binary classification problem, a 50/50 separation. The problem becomes apparent in cases where one class dominates the other, for example in a dataset of 900:100 (class 0:class 1) binary class distribution, classifying all instances as negative results in 90% accuracy score, while not a single example of class 1 has been correctly predicted.

3.2.4.2.3 Precision and Recall

Precision is the ratio of correctly predicted labels out of the total predicted positive labels, i.e. the probability of a positive prediction really being positive, i.e. *P(predicted = observed).* Recall (also called *True Positive Rate*, TPR or *Hit-Rate, Sensitivity*) is the ratio of correctly predicted positives to the total number of positive labels (ground truth) in the data or alternatively, the probability of a positive ground truth, really being predicted as positive.

3.2.4.2.4 F-measure

F-measure, or *F1-score*, also known as the balanced F-measure, is a single scalar value metric summary for the combination of Precision and Recall. It is used in evaluation of accuracy of classification performance in binary problems. F1 is defined as the harmonic mean of precision and recall, equally weighs precision and recall as shown in eq. 2.1:1. As it is based on recall and precision, the F-score only considers the positive predictions. F1-score is generally considered a much more balanced evaluation metric when class imbalance is an issue [52]. A high F-measure score is a good indicator of a good performing classifiers w.r.t. to minority classes.

$$F - measure = \sqrt{\frac{2 * precision * recall}{precision + recall}} \qquad (3.2.1)$$

3.2.4.2.5 Receiver Operating Characteristic (ROC)

A *Receiver Operating Characteristics* (ROC) graph is an increasingly popular visualization technique for evaluation and selection of classifiers based on their measured performance. While not a single valued metric, (a score can be derived from it), the ROC contains useful information related to the specific areas of concern. ROC graphs have originally been used in signal detection theory, and later adapted for medical decision making in diagnostic testing, gaining further popularity. Their increasing use in machine learning is partly due to the growing realization by the data science community that traditional accuracy measures are flawed when dealing with skewed class distributions, which frequently is the case in real world problems. The fact that ROC curves are insensitive to imbalances in class distribution is a very important property. If the proportion of negative vs positive class instances changes in the underlying dataset, the ROC will not change, due to the way it is calculated [53, 54]. A ROC graph is basically a 2D-graph that plots True Positive Rate (TPR, also called Sensitivity or Recall) vs the False Positive Rate (FPR, or 1-Specificity). In this manner, the ROC depicts the trade-off between benefit and cost, calculated by "sliding" the decision threshold.

Fig. 3-7 ROC curve examples: The diagonal represents the random guess. A shows moderate improvement over a random model, while B and C are both much better than A. However, B and C, though different, have an equal AUC-score. In this case, the decision based on the trade-off between sensitivity (y-axis) and FPR (x-axis), must be made.

The diagonal line across the graph (y = x) is the representation of a random classifier, while a line through the point at coordinates (0,1) represents a perfect classification performance. ROC curve is intrinsically used for depicting binary classifier information.

3.2.4.2.6 Area Under the Curve (AUC)

Area Under the Curve (AUC) is nothing other than the area below the ROC curve at unit scale. This makes AUC an aggregated single score metric for performance measure derived from the ROC curve. It is however, not a complete replacement for ROC and should not replace ROC analysis, where fine tuning w.r.t. the target domain of the model is required (see Fig. 3-7). To compare classifier performance by means other than manual inspection, and still draw on the advantages of using a ROC as validation metric, we need a single score value that can easily be compared. The AUC score range is between 0 and 1.0, as it is a part of the unit square area. A random dummy classifier produces a diagonal between (0,0) and (1,1) on the ROC graph and therefore produces a AUC score of 0.5. This implies that no accepted model should have a score of 0.5 or less. AUC score <0.5 can in fact be mirrored over the diagonal by inverting the predictions, negating the classifier. This might contain useful information that has been applied incorrectly [45]. The AUC in all its variants, have been shown to be less correlated with various other metrics, a fact supporting the use of the AUC as a genuinely different and compact measure [55].

3.2.4.2.7 Cohen's Kappa

The *Cohen's Kappa* statistic, (and related *Fleiss' kappa*) is a method for calculating inter-rater reliability[9] based on varying assumptions about the prior distributions that is generally seen as a more robust measurement alternative as compared to simple percentage agreement [56], i.e. accuracy. The score can range from −1 to +1, where 0 represents the level of rater agreement that can be expected from random chance, and 1 represents perfect agreement. A common scale for score interpretation has been suggested in [55], with scores ranging from K<0.20 (poor) to K>0.80 (very good), but it seems like as if no consensus has been reached.

3.2.4.2.8 Matthews Correlation Coefficient

Matthews Correlation Coefficient (MCC) or otherwise known as the ϕ-coefficient is used in machine learning for binary classification performance measurement. MCC takes into account all of the elements of the confusion matrix (TP, TN, FP, FN) and is regarded as a balanced measure applicable to skewed class distributions. It returns a correlation coefficient value between -1 and +1, where +1 denotes a perfect prediction, 0 an random prediction and -1 an inverse prediction. [57]. The MCC metric has obtained increasing interest by the machine learning community. This interest relates to the fact that it fundamentally summarizes the

[9] "In statistics, inter-rater reliability, inter-rater agreement refers to statistical measurements that determine how similar the data collected by different raters are, i.e. a measure of consensus. [56]

confusion matrix (eq. 2.1.2 below) into a single score value, when applied to the binary classification problem [58].

$$\mathrm{MCC} = \frac{\mathrm{TP} * \mathrm{TN} - \mathrm{FP} * \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}} \qquad (3.2.2)$$

It is now commonly accepted as a reference performance metric for imbalanced data in fields such as bioinformatics. In the same study, the case for MCC is further supported as being superior to simple accuracy measure in a skewed class distribution case [58]. In general, MCC is a good compromise of consistency, discriminating power and coherent behaviour under varying conditions.

3.2.4.2.9 Anomaly Score

The *anomaly score* is a HTM-specific scoring metric used exclusively in anomaly detection models [59]. There are two types of anomaly score named in this work: *Anomaly likelihood, log anomaly likelihood* and a*nomaly score*, all of them are fundamentally based on the *raw anomaly score*. The raw metric is just the calculated inverted probability of a prediction producing the same value as the current observation. This is done on every timestep, effectively producing a score for every observation-prediction pair. Anomaly likelihood score is an alternative non-thresholding metric calculated in addition to the raw anomaly score. It is a probabilistic score that evaluates the current state to be anomalous based on a modelled distribution of historic raw anomaly scores from the model [59]. Log anomaly likelihood is just a log-normalized likelihood score. In practice likelihood values may frequently end in the interval [0.99, …, 1.0], so this translates them into more intuitive range where the typical threshold is set at 0.5. This is an (unofficial) adopted practice in the HTM community.

3.2.4.2.10 Geometric mean

The *geometric mean* (G-Mean) is mathematically a type of average, which unlike the arithmetic mean, indicates the central tendency value of a set of numbers by application of the product of individual values. In machine learning, the geometric mean (Eq.3.2.3) is useful as a metric to measure the balance between classification performance over majority and minority classes. Low geometric mean is an indication of a poor accuracy in the classification of the positives, even if the negative cases have been correctly classified [52]. Geometric mean is generally considered a more adequate metric for imbalanced class problems compared to the standard accuracy measure and has therefore been included in our set of performance metrics.

$$GMean = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}}$$

3.2.3 G-mean is calculated using Sensitivity (Recall) and Specificity (True Negative Rate or TNR, see Fig. 3-6).

3.2.4.2.11 Index of Balanced Accuracy

Index of Balanced Accuracy (IBA) metric [61][60] is a weighted average between the specificity and the sensitivity (TPR or Recall), not only taking the overall classification accuracy into account but also attempts to promote the classifier with better results in respect to the positive class (the minority class). High values of IBA are obtained when the accuracies of both classes are high and balanced. IBA is related to the optimized precision measure, which in turn is biased towards the majority class [61]. IBA produces a standard accuracy measure (ACC) when a classifier performs equally well on both classes however, in cases where the classifier leans the advantage to the distribution of the majority class, the IBA will reflect this with a lower score compared to ACC.

3.2.5 Class Imbalance

In the previous sections on algorithms and evaluation methods, we have repeatedly touched upon the subject of *class imbalance* in various forms. The reason for our focus on this area surfaced as the type of data to be used for the analysis became available and it became clear that extreme class imbalance was unavoidable. Class imbalance in a classification task entails one of the classes being strongly underrepresented relative to the other class or classes in a multiclass setting. It turns out that this type of problem poses one of the biggest challenges in data mining [62]. Real world applications such as credit card fraud detection, medical diagnostics and network intrusion detection are typical imbalanced data problems [52].

Frequently in an imbalanced classification problem it is the minority class[10] that poses the target of interest. However, due to the fact that classifiers tend to be biased towards the majority class, most common classification algorithms do not work well in these problems [52]. Additionally, as we have described in the previous section, some metrics are often compounding the problem. Several methods have been suggested for dealing with imbalanced data as pertains to both processing and classification performance evaluation. Applying appropriate evaluation metrics is important [52] we have intentionally included several imbalance-specific metrics in the previous section (3.2.4.2). Consequently, we will in the following sections describe selected resampling methods for modelling of strongly skewed data.

3.2.5.1 Cost-sensitive learning

The term *cost-sensitive learning* entails that misclassification costs are considered during the model training or creation. This is effectively the case where the algorithm can be penalized for prioritizing one class over another, effectively changing the way most classification algorithms calculate the misclassification error cost. There are two subcategories; first, the algorithm has been explicitly designed to evaluate cost sensitive loss functions directly, alternatively there is the cost-sensitive meta learning category [63]. Sampling, weighting, thresholding and ensemble methods are all examples of the second category. There are various theories for setting the appropriate costs/weights and some success has been achieved, however a major drawback is the assumption that the cost is known or easily derived. In real world problems, this is rarely the case and when successful, may lead to overfitting. Several of these techniques are applied in our experiments, specifically weighting, under-sampling and over-sampling.

[10] By convention, the minority class in imbalanced (binary) tasks is defined as the positive class, while the majority class receives the negative label.

3.2.5.1.1 Under-sampling

One solution to the imbalance encountered in strongly dominant majority dataset is to improve the ratio between classes by reducing the number of the majority instances [64]. The argument for doing so is that in an overwhelming majority of closely related instances, removing a portion of the least influential instances will not have significant detrimental effect on classifier effectiveness. This not only improves the class balance, but also has advantages for storage and training time, especially in large applications. Disadvantages of under-sampling centre around the fact that data instances are removed from the training set, which results in less accurate classifiers. There are various approaches to under-sampling; Random, Clustering based, Nearest Neighbour, etc. [60]. NN approach such as Tomek-links, examine the neighbourhood instance space, compare pairs of instances of opposite classes that are their own nearest neighbours and remove these. By eliminating the closest opposite-class neighbours, the minority class region in the instance space becomes more distinct [64].

3.2.5.1.2 Over-sampling

In over-sampling, the main idea is to influence the class imbalance by increasing the number of positive instances. The most popular advanced over-sampling method is SMOTE (*Synthetic Minority Oversampling TEchnique*) [65]. SMOTE, in its many variants attempt to create new data instances from an existing set and differ from each other in the way that this is done. In general, specific similarity measures are used to define and create instances that would (theoretically) be classified as part of the positive (minority) class. This is also called the generative oversampling, as it generates new instances based on the learned distribution of existing data. Contrary to the under-sampling technique, over-sampling does not remove information from the data, no instances are ever removed from training the classifier. The drawback is that in the frequent case where imbalance goes hand in hand with inadequate number of minority samples, the learned distribution estimate is potentially inaccurate and variance artificially lowered [66].

4 Methods

This chapter describes the methods that were used in order to answer the research questions from section 1.4. In the next subsection, we describe classification specific methods and HTM, followed by section on selected preprocessing tasks such as segmentation via sliding windows, sleep period recognition and domain feature extraction.

Traditionally, to run machine learning experiments, the main requirement is the availability of relevant data. Often the process of data collection is initiated after some idea or research question has been set out to which this data will potentially provide an answer. As mentioned earlier, this work is concentrated on already existing, labelled data. It was decided to use the Data science methods as observed in the reviewed research in section 3.1. More specifically the methods observed in [43, 46, 80, 86] and others. These are all classification-oriented. Consequently, we deal with specifics of HTM in the next section. The basic steps in the scientific method for data mining then become:

- Begin with the research question
- Leverage and explore the data (preprocessing)
- Extract & create features (preprocessing)
- Explore parameters, create model(s) (training, performance evaluation)
- Discuss the results and draw insights
- Conclude and point out shortcomings

4.1 Classification

With the research questions already having been set, we move on to the data exploration and preprocessing steps. We advise the reader to study the diagram (Fig. 4-1) describing the detailed steps of the process used in the experiment phase as we briefly describe it below.

Exploration of the data is done to gain early knowledge of the possible manual preprocessing actions that might be required. Having good knowledge of the data and its domain can be helpful in the often-creative feature extraction process. Furthermore, visual clues such as artefacts, irregularities, amount of noise in time series signal or missing values in tabular data may help in selecting the right type of training methods and in avoiding time consuming mistakes.

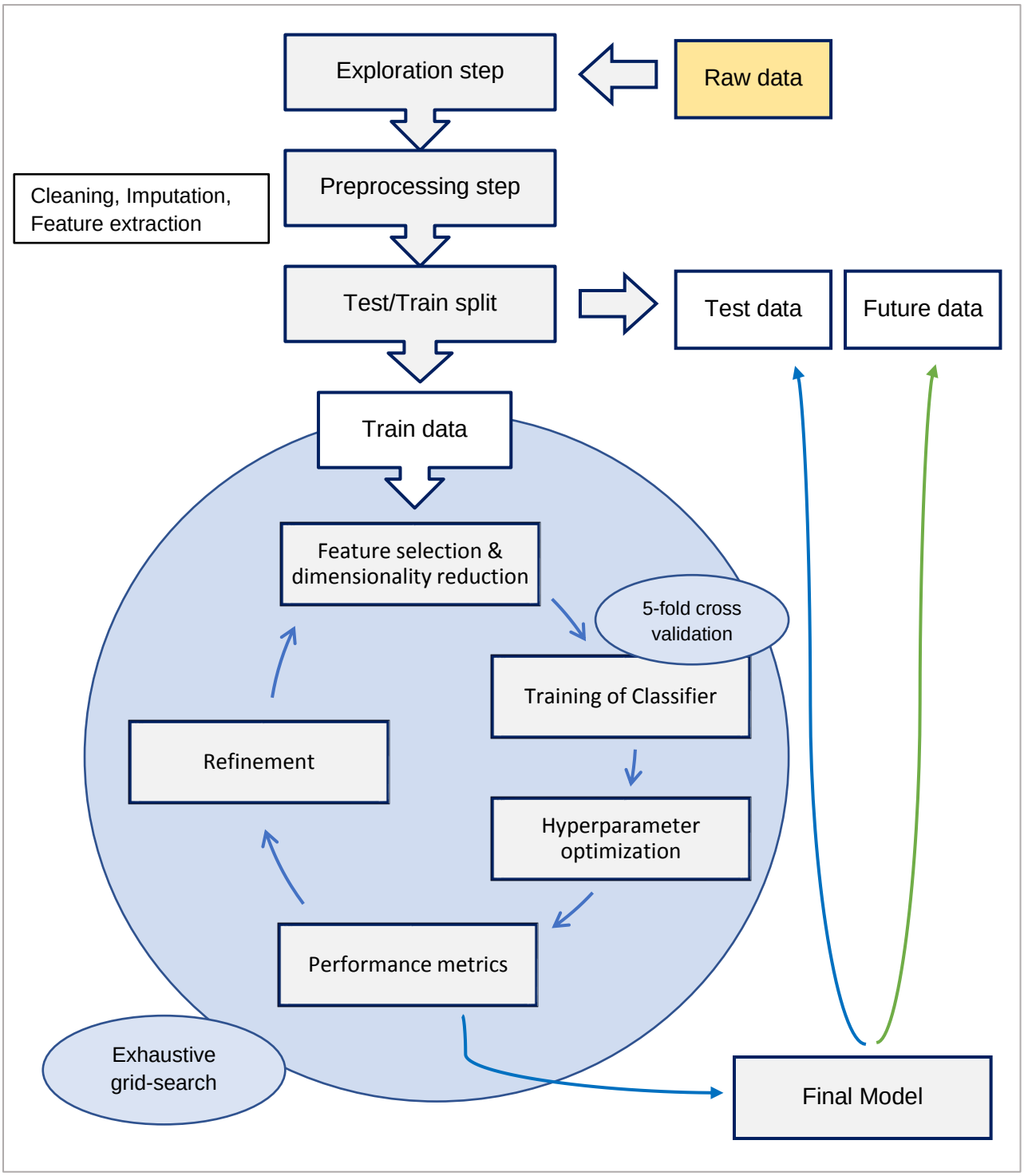

Fig. 4-1 Supervised learning process model used in the classification experiments. Grid-search with 5-fold cross validation was used to get the final model which was then scored on the hold-out test data. This ensures a clear split between test, train and validation sets.

Pre-processing involves a multitude of tasks with the common aim of making the information in the data more accessible for the learning algorithm. These tasks include cleaning, restructuring, imputing (removing or replacing invalid values), various transformations (numeric to categorical, categorical to binary, text conversion etc.), standardisation and normalization of numerical values [42]. Some are necessary: e.g. MLP requires normalized numerical input [24], while distance-based kNN will produce wrong results when feature attributes have different scales.

For all experiments, the data has been converted from supplied AWD format to a timestamped CSV file with additional migraine attacks annotations added according to separate documentation provided with the data. A binary attribute column is created from time information of the migraine attack. This attribute was then used (with modification in some experiments, see below) as a target variable for the classifiers. For designated experiments, the data was additionally truncated where the actigraph has been unequipped, while remaining activated creating a very regular artefact in the signal. This is reflected by the "*exs*" setting in the individual experiment settings as shown in 8Appendix C . Finally, segmentation into windows and feature extraction are implicit parts of the preprocessing tasks. Both are described in detail in the sections below.

Ready segmented and prepared data, was now split into a training and test data set. We used a 70/30 split, meaning that 70% of the data was randomly selected to be used for training of the model, while 30% went to the hold-out test set, to be used in final evaluation. The random generator seed value for the split was set constant for all experiments (*seed=505*) to enforce reproducibility.

The experiments were run using a pipeline[11] which included 10 selected classification algorithms as described in section 3.2.1. Each of the selected algorithms received a limited set of hyper-parameters as shown in 8Appendix A . These sets were common to all experiment runs. Hyper-parameter tuning of a model was carried out using an exhaustive exploration of these subsets of the space of possible hyper-parameter configurations. The objective was to find hyper-parameter values which lead to optimal classification performance. This type of search is also called a grid-search, as the arrays of parameters form a multi-dimensional grid.

Within the grid-search loop, a secondary, optional preprocessing step consisting of feature selection and/or dimensionality reduction (see section 3.2.3.2) was undertaken, as both factors may noticeably influence the training process. This is reflected by the attributes (for complete overview, see 8Appendix C) included in the result tables.

Cross validation was used for estimation of model prediction performance during training. We have decided to use the K-fold cross validation method, (with $k = 5$) for all the classification experiments. Additionally, we used random sampling with stratification, meaning that for every

[11] To aid in the combined process of finding the best combinations of parameters, training, validation and testing, it is highly recommended to automate it. In the Data science community, this is also called creating a pipeline.

test fold the class distribution of instances will retain a similar ratio to the original train set distribution. This is of particular importance when dealing with imbalanced data sets, such as our migraine data (see section 2).

Notably, grid-search hyper-parameter optimization with cross-validation is a very time and resource consuming process with some experiments taking up to 1300 minutes, i.e. about 20+ hours.[12] Because of this we limited the hyper-parameter search space as well as the number of cross-validation iterations.

Class imbalance is a specific and serious challenge that had to be dealt with in our work. Several methods were taken into consideration including under-sampling, over-sampling and combinations with proposed imbalance-oriented metrics (see section 3.2.5 and 3.2.4.2). To start with we introduced our own simple over-sampling method (SOS) which involves labelling windows adjacent to the original target window[13] as valid targets. This was done at the preprocessing stage when creating window instances, by positively labelling windows within a certain distance (measured in timesteps) from the true positive. The advantage of this method is that the synthetic new instances have the closest similarity to the original, when measured using typical distance measures. (see section 2.1). Naturally, increasing the step distance decreases similarity and the best settings should be set by empirical testing.

In addition to our SOS method we decided to use the popular over-sampling technique SMOTE, described in section 3.2.5.1.2. SMOTE creates synthetic instances based on the instance attributes (in comparison, our oversampling method labelled an existing instance based exclusively on sequence similarity, before instance attributes were calculated). Consequently, for under-sampling we utilized another, though less known method called Cluster-Centroids, an implementation of which is found in the "Imblearn"-package [60]. The selection of the above two methods was based on execution of preliminary tests designed to provide the best suited out of a set of possible candidates (see Fig. 4-2 and Fig. 4-3).

[12] One experiment consists of a full grid search for each of the 10 algorithms, including 5-fold cross validation and final performance measured on the hold-out test set.
[13] A window related to the sequence containing the target label.

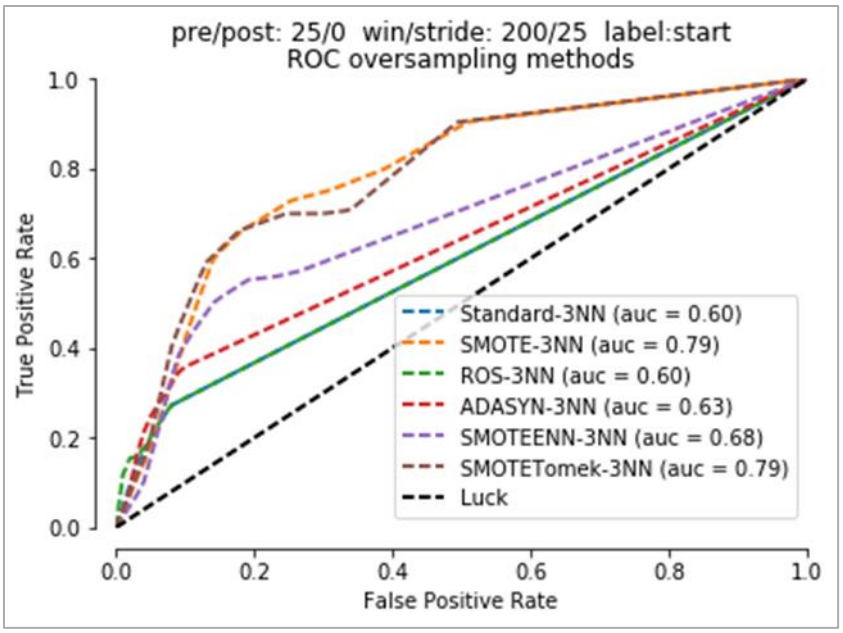

Fig. 4-2: ROC curve comparing multiple oversampling methods. The underlying classifier for this comparison was kNN with k=3 and one of the migraine datasets.

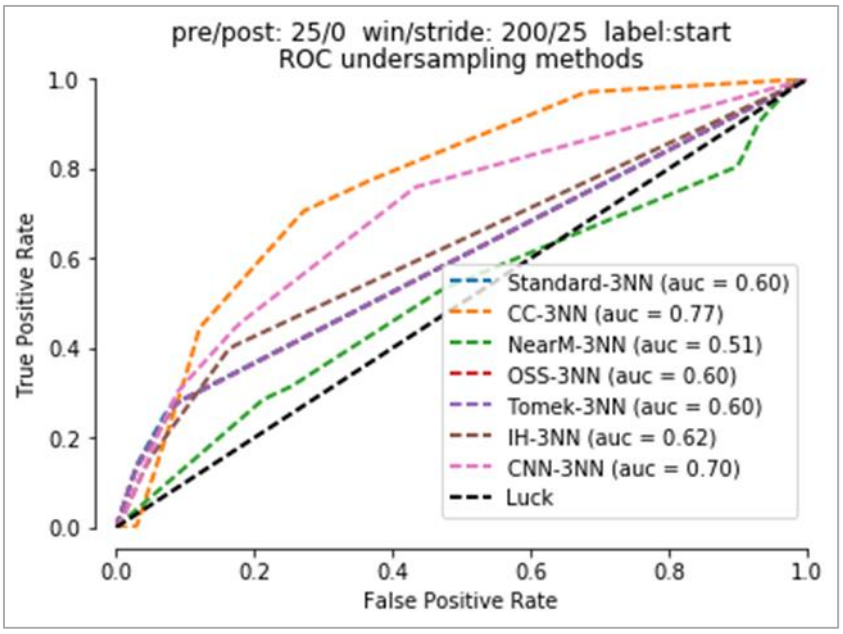

Fig. 4-3: ROC curve comparing multiple under-sampling implementations. The underlying classifier for this comparison was kNN with k=3 and one of the migraine datasets.

Finally, the choice of metrics used for classifier performance evaluation play an important role. Although we included all of the metrics described in section 3.2.4.2, we had to decide which single metric would be used for training the model. The model produced is dependent on the metric used, e.g. using accuracy could be a mistake as described in section 3.2.5 on class

imbalance. Based on the combined knowledge gained from examination of related literature, it was decided that MCC will be the metric of choice as it has the desired characteristics [53] and is more frequently used (unlike IBA, Geometric mean or even the Kappa statistic). Using a more popular metric, should ease the reproducibility and promote verification of results by 3rd parties.

4.2 HTM

Applying unsupervised learning methods is generally simpler than classification. This is also true for our HTM experiments.

While exploration is a very useful step, it was not required because we did not need to extract features. Basic preprocessing and indexing were performed to supply correct timestamps to the algorithm at runtime. Header information from supplied AWT files was removed except the monitoring date/time from which the timestamp information was extrapolated and the samples timestamped accordingly row by row. This is required in experiments using the Date encoder, which will allow the model to relate certain time periods with certain temporal patterns (sequences over time). A simple overview of the HTM experiment structure is given in Fig. 4-4.

The process "train" period in the diagram depicts the process of HTM encountering data with no prior history, i.e. the model was allowed to learn a certain number of timesteps while we ignored its predictions. This is how the model learns what is normal. We allowed three days (4320 timesteps) for this process before the predictions from the model were accepted.

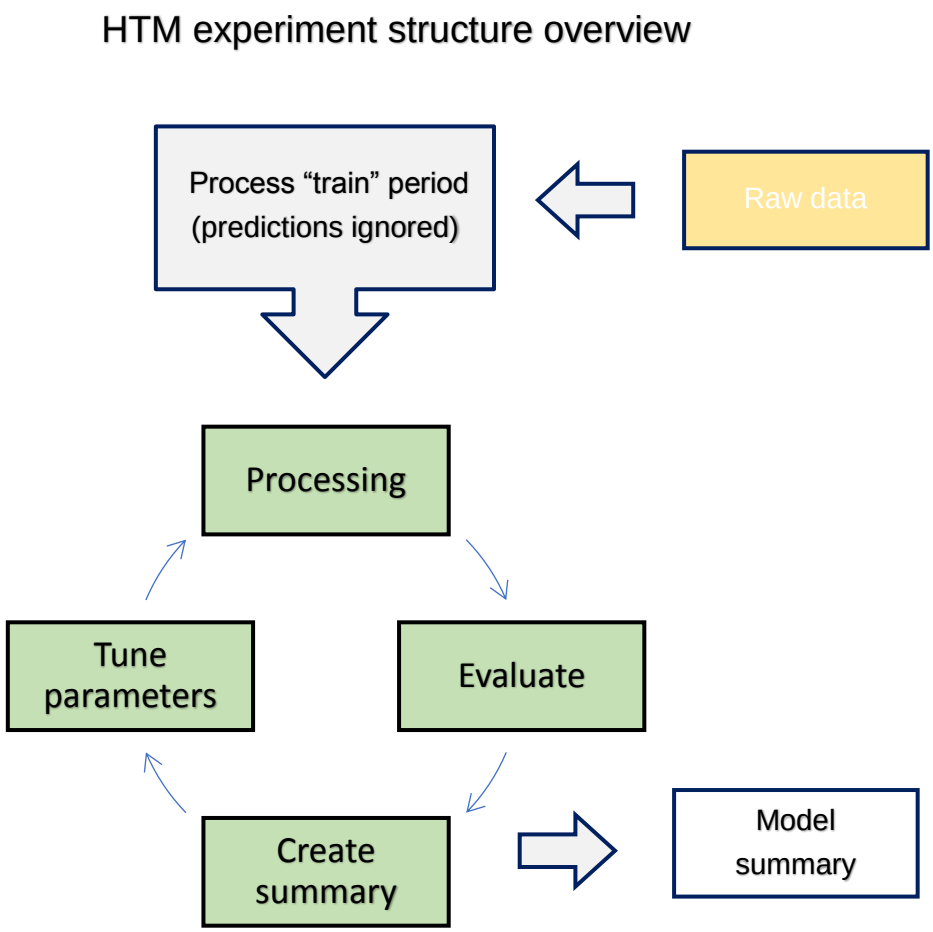

Fig. 4-4: Representation of the HTM experiment structure.

Target labels (migraine attack timestamps) were supplied in the data, but were never used as input to the model. Following the “train” period, the remaining raw data was fed to the model in a continuous sequential fashion. The anomaly scores were logged for each step and additionally, a separate summary file was created better suited for presenting the results. It is during the creation of this summary file that the target labels were used to calculate accuracy of model predictions. Next, the parameters of the model were tuned, i.e. as in Grid-search described in the classification section above, the parameter combinations were selected from supplied sets of values. Finally, the next model was ready to be processed independently with the new set of parameters.

4.3 Pre-processing

This section describes in more detail the task-specific preprocessing and preparation that was done prior to the classification experiments.

4.3.1 Sliding window approach

Crucial and powerful, the sliding window technique has been successfully applied to many classification tasks involving time series data. The idea is to create a segment of a certain (adjustable) size and slide it over the entire time series sequence, consequently shifting the elements of the time series through the window. At each step (also called a stride), the window content is processed, performing calculations, such as various statistics, over the windowed sequence (Fig. 4-5). Given *N=number of data points*, then *N/stride* is the number of windows created with *stride* points between consecutive windows' start indexes. The systematic mapping of features (the feature vector) thus calculated can subsequently be labelled, creating a training instance which can be used to train a classifier.

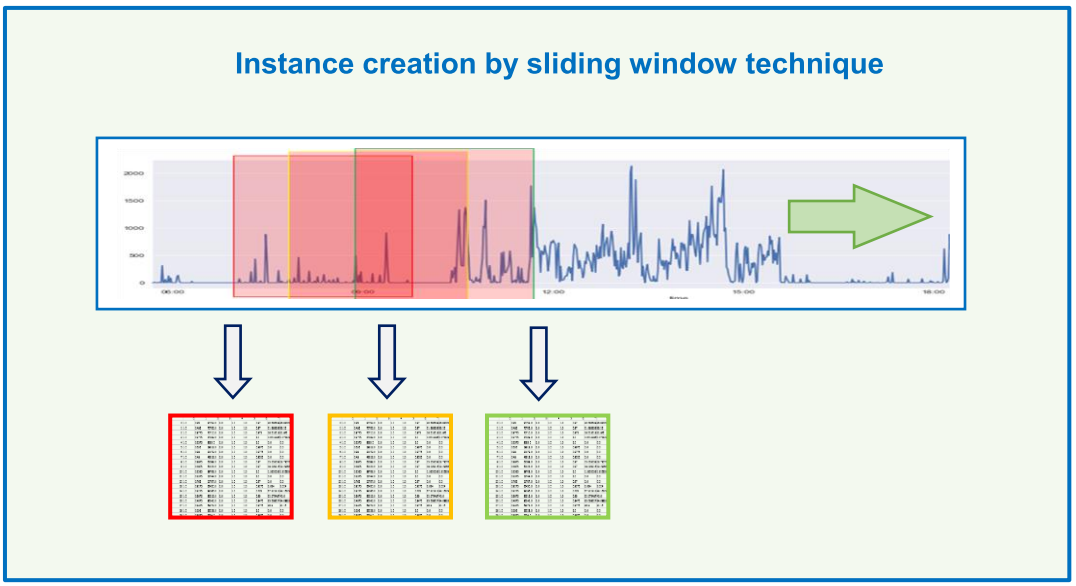

Fig. 4-5: Simplified diagram presentation of the process of feature vector creation using overlapping sliding windows technique. Features calculation (depicted by vertical arrows) is done for each segment bounded by the window as it slides along (green arrow) the sequence data. Each of the produced (bottom windows) is now a labelled instance of the training and/or test sets.

All our classification experiments utilized the sliding window approach prior to execution. The importance of the two parameters, i.e. size and stride cannot be understated, as it directly translates to the amount of time units of sensor data required to make a prediction. The stride

parameter in effect states the degree of overlap between the subsequent windows. An overlap was assumed to soften the detrimental effect of slicing up the time series into independent chunks, as data points in a sequence are seldom independent of each other. The overlap attempts to preserve some of this information.

4.3.2 Active sleep period recognition

Sleep quality is an important factor in affective illness. Consequently, detecting sleep periods has traditionally been of importance in studies involving actigraphy [7, 75]. Although we did not focus on sleep patterns in this study, it is still of importance to be able to differentiate sleep periods from active periods. The main reason is the possibility for more accurate domain feature extraction, due to the ability to discriminate between periods of continuous time, especially when time of day information is not being utilised in feature extraction. Secondly, partitioning the main dataset into night and day potentially narrows down the size of the negative examples as no migraine attacks have been registered during night periods in any of the supplied datasets[14].

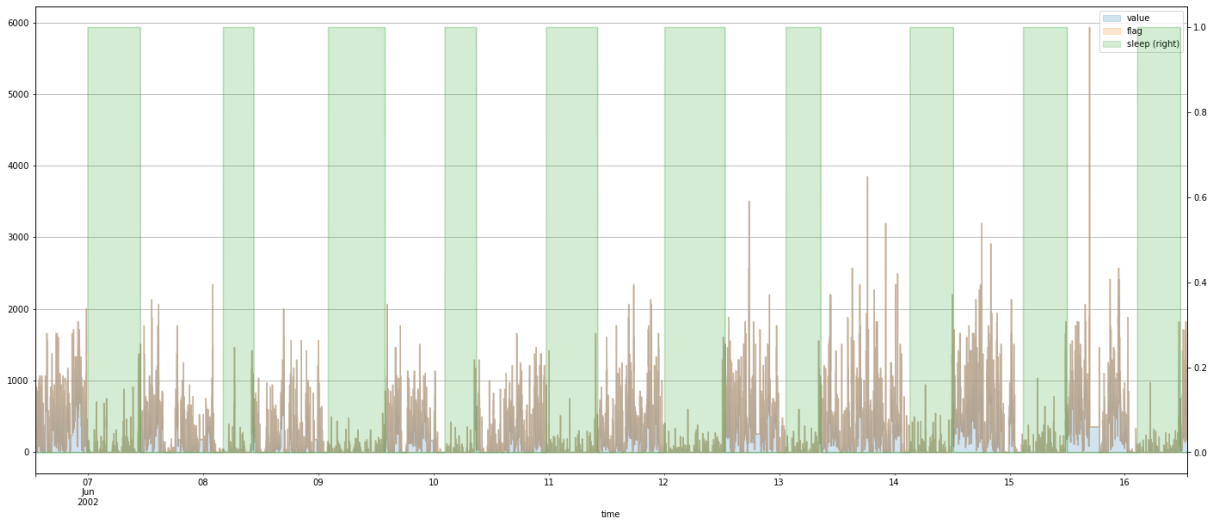

Fig. 4-6: Example of circadian cycles for one patient with visually segmented night and day periods as detected by the sleep recognition algorithm.

[14] Although it is a factual and useful observation, we do not at any point assume this to be true for any future observations.

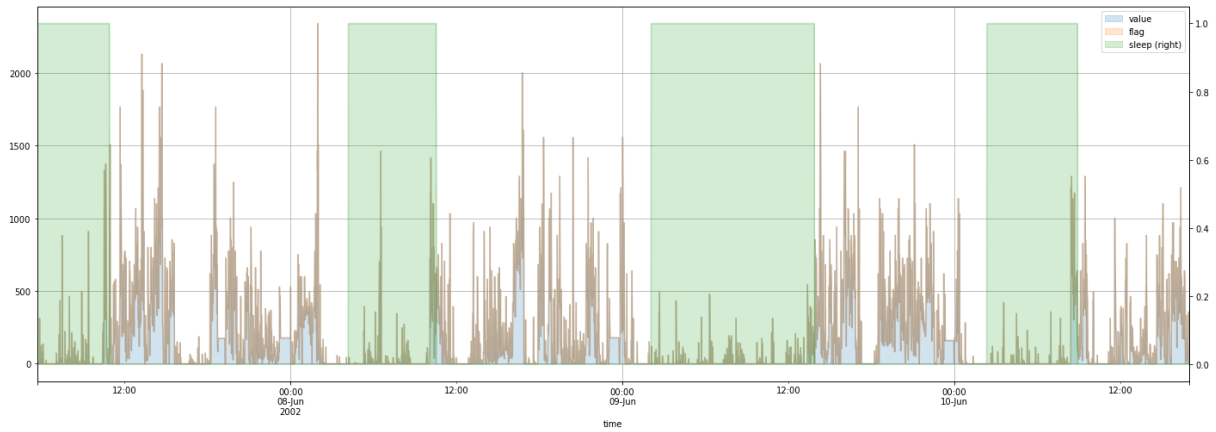

Fig. 4-7: A close-up of the segmented night/day data from Fig. 4-6 above.

The algorithm and parameters used in the following sleep detection algorithm were based on study by Nová, Albert, et.al. [5], with modifications:

As a preprocessing step the signal is smoothed by a median filter, window width N=21. Three windows of length (lev1 = 10, lev2 = 120, lev3 = 30) is initialized. Next, the algorithm finds lev1 − 1 non-zero samples before the i-th-sample and lev1 zero valued samples following ith-position. The resulting list of points is used in the following step where the algorithm selects the index starting a window of lev2 samples that contain 75% zero values. This index determines the beginning of a sleep period. End of sleep is defined in a similar fashion as the index with lev3 consecutive non-zero values. Modifications were made in the filter parameter (N=21 vs N=20, thus encouraging integer result) and more importantly, the critical sleep/waking transition points were accordingly shifted on the unfiltered data to counter the effect of sliding window returning indexes offset by the window size. The result of this should give more accurate inflection points on unfiltered signal.

4.3.3 Domain features

The following features were created based on the literature review: skewness delta, kurtosis delta, prior night sleep quality and time of day. The motivation and practical details are described in the following subsections.

4.3.3.1 Skewness & kurtosis delta

In the original study of Fasmer et.al. [70], the skewness/kurtosis showed a notably significant difference from the distribution in the window just prior to the event and the window starting with the id of the event. We will call the 1st window "before" and the 2nd window "during" for simplification of terms. The two windows were not overlapping i.e. "before" ends before the "during" which must start at the point of labelled migraine attack. A secondary requirement was that each of these windows had a minimum of 50 data points uninterrupted by sequences of 0-activity of more than two consecutive 0 values. A number of examples could not fulfil this requirement in the original study. In our experiment, we decided to relax this requirement so as not to decrease the already small positive sample population.

The feature created from this description would have to consider the sliding window size used, which in turn could mean that a) if the window was smaller than required activity period, no calculation could be made, or b) if the window is bigger, it should be adjusted in size to the length of the period it is calculated for. In the latter case, the complexity of using irregular windows and deciding where to break up the window was deemed needlessly high. While we could map out the skewness windows as a preprocessing step over the whole dataset, this would not present a realistic solution in a real application with continuously received data sequences. The optimal alternative would be to find a relation between an earlier occurring, prior feature and a subsequent attack later in time.

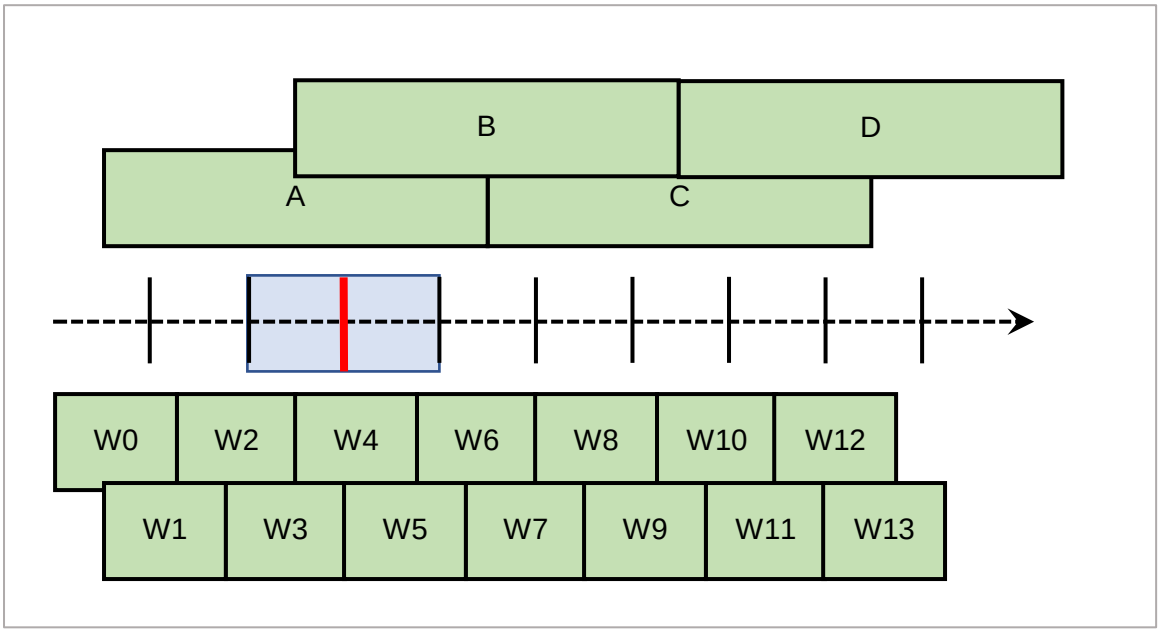

Fig. 4-8: Example diagram showing 2 different sized sliding windows (A-D & W0-W13) encountering an attack point. The shaded area before and after the point are the before and after periods used to calculate the skewness/kurtosis delta feature. In this example window A may hold the calculation result, but it will be B that will be assigned with the target label (it starts closer to the attack point). In case windows are sized and overlapped as below, w5 could be assigned the label, but the calculation would have to be done based on past windows, because of the size of the overlap.

When looking at Fig. 4-8, assume that the point boundary w3/w5 (shown with the red vertical line) is our migraine attack point. The calculated feature of kurtosis/skew delta for the before/during periods would be naturally attached to the feature vector of w5. This is because w5 completely contains the end of the during period, so the calculation is possible. The target label is set to the window with starting point closest to the point of attack. In this instance w5 has both the target label and the valid result of the calculated skewness/kurtosis feature. However, in case a sliding window of size A/B/C, the A window will contain the relevant feature value though incorrect (as A overlaps before & during sequence) and B will contain the target label. In other words, it is the first completely containing window after the target boundary, that will contain the feature. Another problem arises in case target is set to w4 (assuming w4 begins closer to attack point): we need to decide which of the windows should be labelled with the target class label. If we set target label to w4 as it contains the target, w5 will (correctly) contain the skewness feature for before/during delta, which the window with the target label (i.e. w4) should have received.

Finally, it was decided that we would simplify this feature calculation by applying lagged skewness and kurtosis statistics of previous windows to each individual instance. While low stride values relative to the window may create overlaps, past window information is not lost. This also allows for much more flexibility w.r.t. the window parameters, i.e. keeping them fixed. Furthermore, in a preliminary experiment we visually explored how discriminative a feature would be obtained from this alone (Table 4-1). Apparently, the result suggests that in most instances, the skewness comparison feature has questionable relation to a migraine event

in context of all potential before/during transitions. Although the migraine attacks indeed show high frequency of low value – high value pairs, when we look at the overall picture we see there seems to be no consistent rule. This indicates that using the difference as a feature alone probably adds only weak discriminative power.

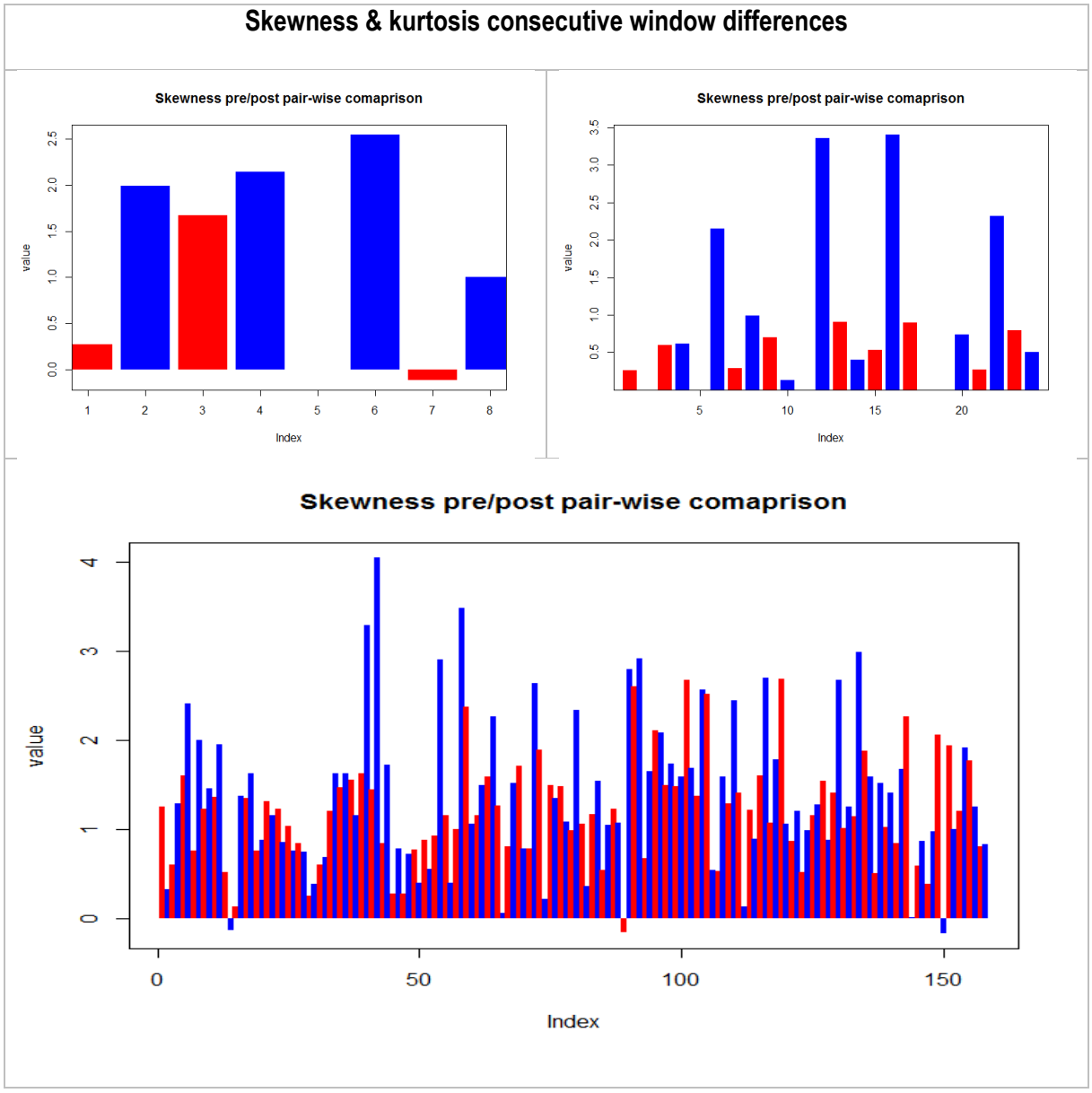

Table 4-1: Skewness pre/post windows comparison based on the dynamic windows described in [70]*. Y-axis depicts the skewness value obtained. Red bars represent "before" windows, blue bars represent the "during" window (after the timestamp of attack). Top row: example from two migraine patients, filtered out for labelled attacks only: showing relatively consistent higher values of the "during" window. Bottom: All consecutive dynamic-size windows (min. size=50, max=64) from one of the above patients.*

4.3.3.2 Prior night sleep quality

Prior night sleep quality is a quantitative measurement of total activity during the period of sleep preceding the current timestep of activity. Creation of this feature was inspired by the study "Sleep and migraine: An Actigraphic study" by Bruni et.al. [75]. The sleep period as calculated by the active sleep recognition algorithm in section 4.3.2 over is used for the calculation of mean or alternatively the frequency of occurrences above a pre-specified threshold value. It was decided to set this threshold at $T = 500$, although the best value to use here is an open issue, that would require further experiments. We did however compare the performance of the mean-method vs the threshold method in a few simple exploratory experiments. Results are shown in Fig. 4-9 and might need some explanation: In general, we would want the red bars to be as short as possible, meaning that on the day of attack the night prior to it was measured with low activity. Low activity measures during the night are assumed to be a sign of relaxed and therefore good quality sleep. Additionally, the red bars should be on average shorter than black bars. Naturally, this feature is highly dependent on the sleep period being estimated correctly. The 3rd example has highest frequency of migraine days vs normal days and is therefore potentially a poor example.

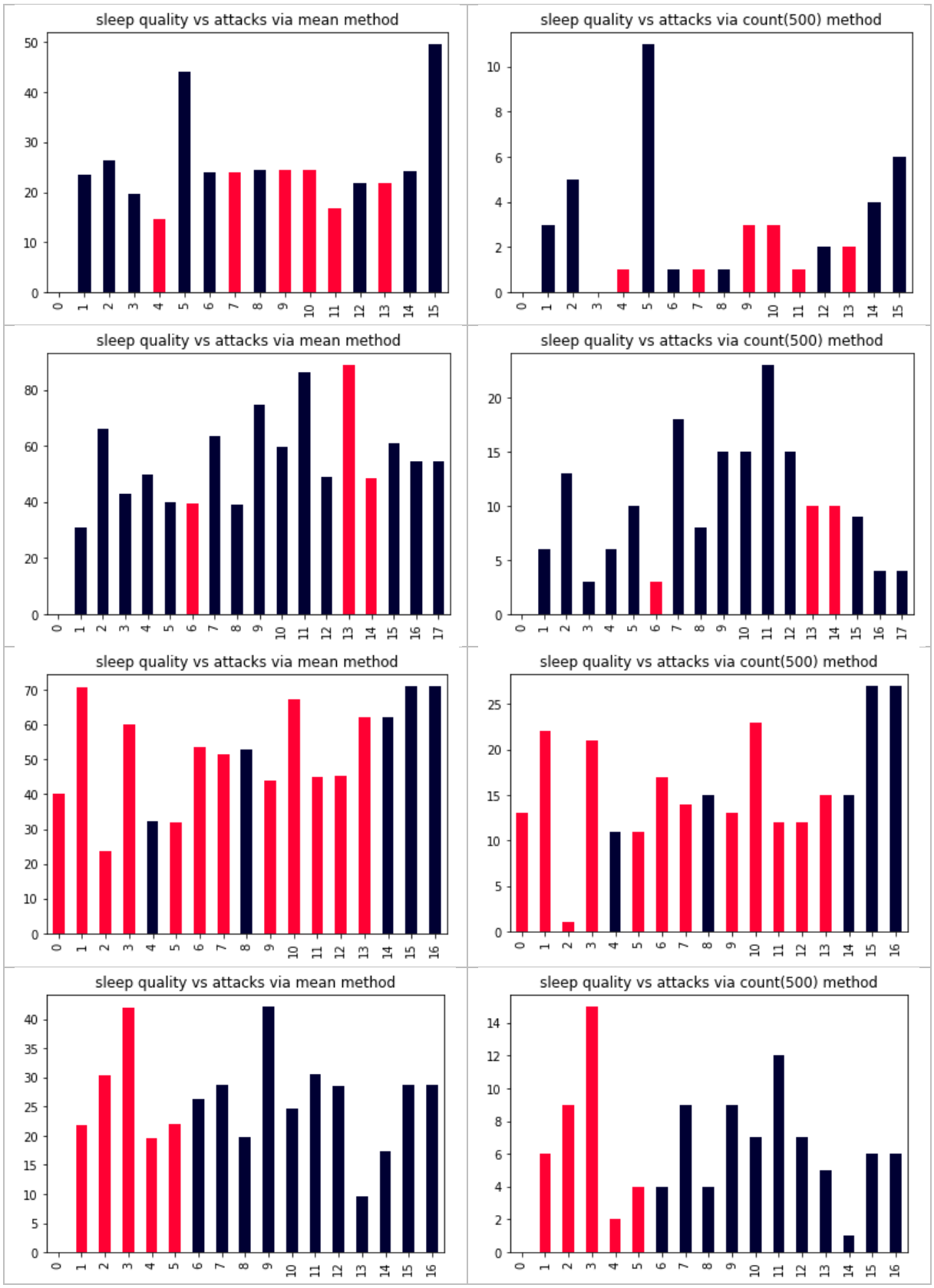

Fig. 4-9: Night sleep quality measurement method comparison: Red bars depict days with migraine attacks, black bars depict days with no attacks. The magnitude of the bar represents the corresponding prior night sleep quality score value for that day. Left column shows relative scores using the mean-method, right column shows relative scores using the count (T=500), method.

4.3.3.3 Time of Day

This feature was largely inspired by the Date encoder for the HTM algorithm (section 3.2.1.8 over). The idea that a relationship exists between circadian cycles and factors influencing physiological and mental states is not new [5, 7, 78]. During one visual inspection of the migraine data, it was discovered that in all four migraine patients over 90% of attacks were registered during the afternoon and evening hours. Naturally, the small sample size of the population we operate with is a strong argument against drawing any major conclusions. However, this observation was something that we believe can add to the discriminative power of the extracted features, and in the end, it is the classifier that makes the decision whether this is a useful feature to consider. The implementation was very simple: the 24-hour, day/night period was divide into 3 equal, 8 hour segments, creating a categorical variable with 3 possible (numerical) values. The time segments were chosen as follows: { (05:00, 12:59), (13:00, 20:59), (21:00, 04.59) }.

5 Experiments & Results

In this chapter, we present detailed description of implemented experiments and the results obtained. It consists of two subsections covering the two different approaches used: unsupervised learning using HTM, followed by supervised learning using selected classification methods. In each case additional, method-specific preparation and preprocessing might be necessary, in which case further details are included.

5.1 HTM experiments

The idea to use anomaly detection came from the assumption that regular sensor readings taken over a prolonged period will develop similar patterns, effectively creating "the norm". An adverse event of some kind that is reflected by changes in the underlying sensor readings would then considered abnormal in context of historic readings, "the norm". Assuming some meaningful change has occurred, we would wish to find it and optimally identify it with certain type of events. While the latter part is purely a classification problem, the part of finding the abnormal pattern before we know what to look for suits an unsupervised learning task.
The adverse event becomes the migraine attack, and we want to monitor the signal (before and during) to discover any significant correlations with potential abnormalities.

5.1.1 Practical issues

We begin with additional details of implementation and practical considerations for the HTM experiments. Numenta's implementation of HTM theory is written in C++ and Python 2.7.x. Underlying libraries are organized under the name of Nupic (Numenta Platform for Intelligent Computing). Java implementation has been ported from Nupic by community members under the name of HTM-Java. Nupic is distributed under a variant of AGPLv3 open source license[15]. For parties who are unable to use the AGPLv3 license a separate, trial license without commercial rights[16] has been supplied. The libraries are freely distributable for non-profit research purposes as stated on the Numenta homepage.

[15] As described at http://numenta.org/licenses/
[16] Trial license can be found at http://numenta.org/licenses/trial/

HTM-Java is not an officially supported release at the time of writing. There are issues with the codebase, so even though it was tempting to use the Java version, the decision was made to make use of the more stable and officially supported Python version in the beginning. Installation was slightly problematic: it has been tested under specific versions of Linux, there were several installation attempts on different versions of Ubuntu from 14.04LTS through 15.10 to 16.04, all with some issues, or failing unit tests. Finally, an acceptable solution was found, though 18 of the 800+ unit tests were still failing for unclear reasons. The core functionality seemed to work properly on example applications so this installation was accepted. Further problems were encountered a few months into the project after an official update made several breaking changes. The created models had compatibility issues with the output of the swarming scripts. We must add that as Nupic and HTM is still work in progress with no official release version available (as of May 2017), such problems should be expected.

For all experiments, the data has been converted from supplied AWD format to a timestamped CSV file with additional migraine attacks annotations added according to the documentation provided with the data. The files have not been truncated or otherwise additionally modified.

5.1.2 Regular signal

As an important first step, we need to know that the algorithm functions as intended and to gauge its effectiveness, we start with a simple sine wave experiment. HTM has been reported to be effective for processing continuous, regular data streams, this is what we will use as a reality check. A simple sine wave is generated and the points stored in the format ready to be processed by the model. We let the algorithm learn the pattern for X full cycles (or Y points), after which the detection is activated and we measure how it scores the rest of the sequence. The assumption is that anomaly scores for the testing sequence should be low, as the distribution of data has been seen previously. Simple sine wave signal has been created and the amplitude scaled accordingly to the values found in the migraine actigraph data, i.e. range (*min. = 0.0, max = 4000.0*). The signal is generated over 5000 1-minute timesteps, following the same timestamp format as the processed actigraph dataset. The training period is set to 1000 points, during which anomaly scores are ignored. A scalar encoder is used for the model with complete parameter settings as shown in 8Appendix B . For delta encoding, the effect is achieved by feeding the model differences of consecutive pairs of values and modification of valid value ranges of the model parameters, which are otherwise exact duplicates. Under execution, the model outputs a raw anomaly score (section 3.2.4.2.9) for each step[17], depending on how confident it is that the sequence leading up to this point has similarities to sequences learned earlier. HTM produces probabilities for its output predictions. The probability distributions are used to score the incoming data to produce an anomaly score, which is high (i.e. anomalous) for sequences before unseen and low for sequences that have been encountered before or bearing enough resemblance.

[17] With single-step prediction model, which is used by default unless otherwise specified.

The anomaly score ranges from [0.0 to 1.0], where a score of 1.0 entails that the current sequence is completely new and unrecognizable, and therefore, an anomaly. Consequently, a value converging on zero, translates to increasingly recognizable patterns.

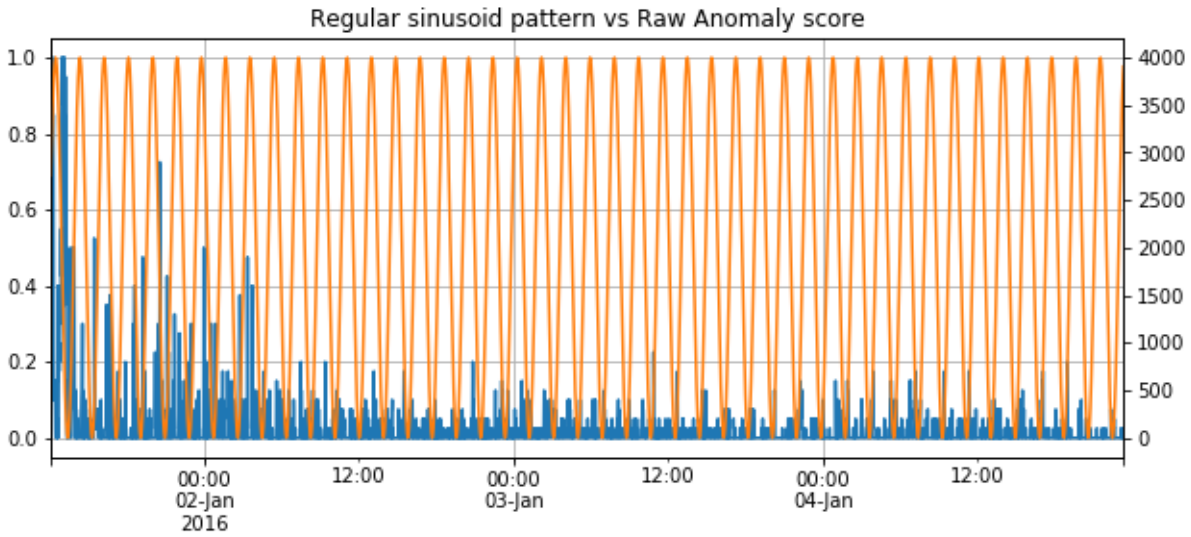

Fig. 5-1: Regular sinusoid pattern (orange) vs raw anomaly score (blue vertical bars), complete plot. Traditionally scores > 0.9 are flagged as anomalies. The anomaly score noticeably subsides with time as the pattern is repeated.

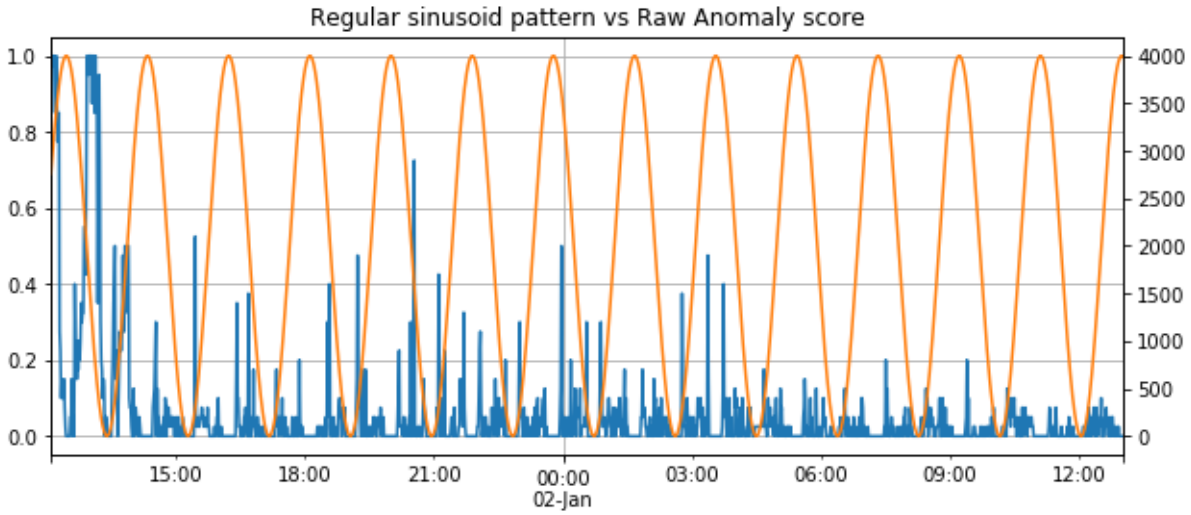

Fig. 5-2: Zoomed in on the first 1500 data points of Fig. 5-1. Several strong anomalies (blue vertical bars) are recorded early in the learning phase, consistent with the model not having seen this pattern earlier. This is normal and expected behaviour.

In Fig. 5-1 and Fig. 5-2 we see how the model learns the pattern, the anomaly score functioning as an effective (inverted) indicator of confidence in current prediction. It is important to note that that the anomaly prediction is done in real-time, as the signal arrives, while the plot shows the complete history. From the above we can see that the model is able to learn the regular

pattern, resulting in very low anomaly scores as time progresses. This is true for both encoding types (see Fig. 5-3 & Fig. 5-4 for delta encoded alternatives).

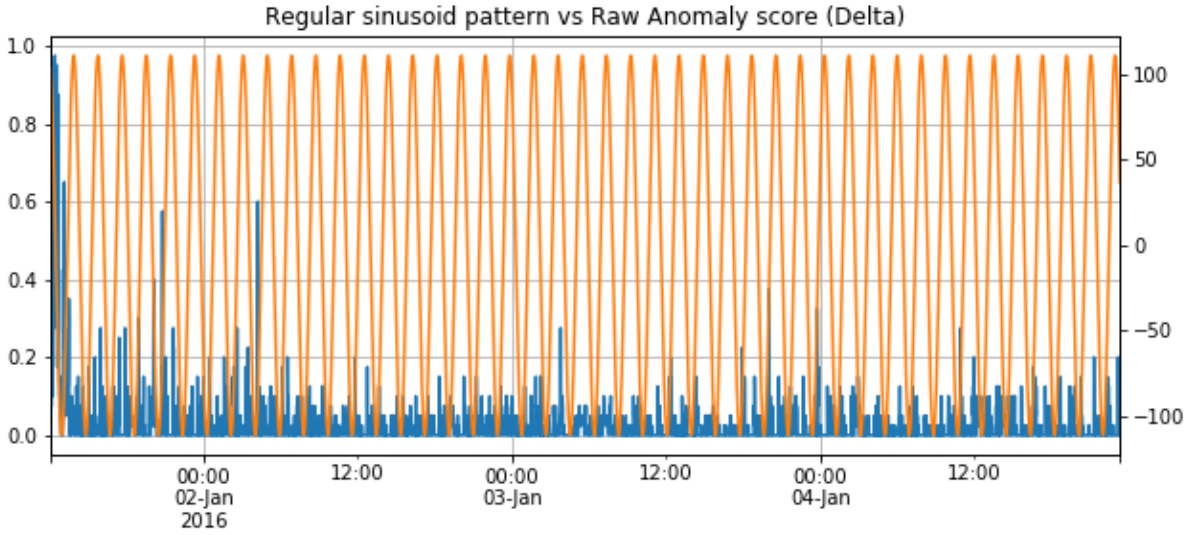

Fig. 5-3: Regular sinusoid pattern deltas (orange) vs raw anomaly score (blue vertical bars), complete plot, using the delta encoding technique. The anomaly score seems to adapt quicker in the early stages when compared to Fig. 5-1.

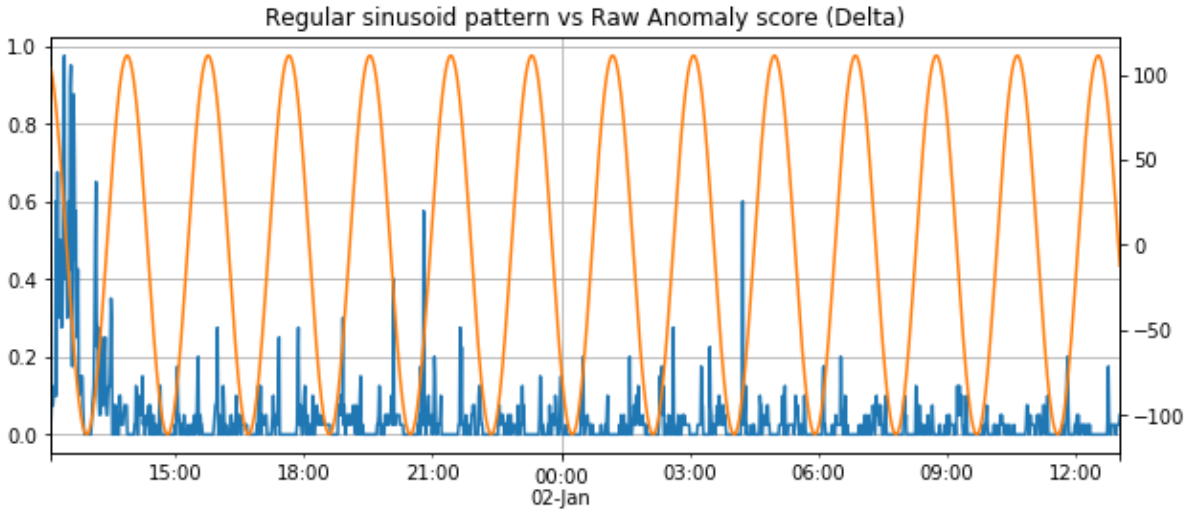

Fig. 5-4: Zoomed in first 1500 data points of Fig. 5-3. (The anomaly score scale is on the left axis, the effective delta value is the right axis)

To measure which model outputs more confident predictions, we counted and calculated the mean totals of anomaly scores over a pre-set noise threshold, which we set to 0.3. A lower threshold will pick up less significant variations, which at these empirically low levels can be considered as noise. However, we include the 0-threshold (total) score for completeness in Table 5-1:

	t > 0.3 (count)	mean	total points	total mean
Standard encoding	11	0.40226	4498	0.01677
Delta encoding	4	0.46875	4498	0.01762

Table 5-1 A summary of anomaly scores for standard value encoding and delta encoding after 500-point training margin. Threshold (t) breach counts, mean, followed by total count and total mean. Standard encoding is more prone to alerts with 11 counts, while delta seems to produce fewer but with higher confidence.

In the next step, we added two anomalies to the existing sequence, and again used both models in the comparison (Fig. 5-5 and Fig. 5-6). The two anomalies are purposefully created in different ways: the first emulates the "dip" of the curve, but bottoms out about 10% higher than usual. The second anomaly is a rough "slide" of strength +/-600 over 10 time steps.

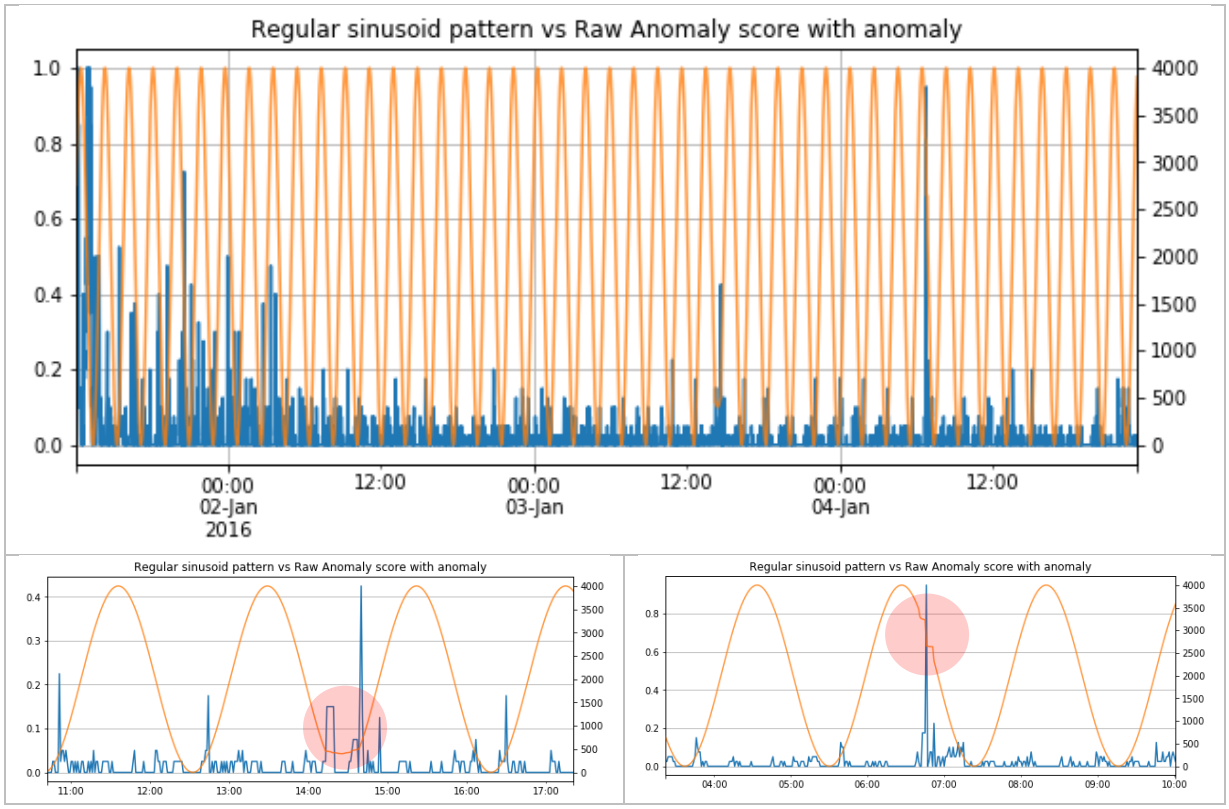

Fig. 5-5: The same sinusoid pattern with 2 artificially added anomalies. (highlighted, just before 15:00pm Jan 3 and 07:00am Jan 04). The subplots show close-ups of the anomalies. Note: the first anomaly would normally not be flagged with the achieved score of just above 0.4.

As we can read from the plots, the delta-model is visibly more sensitive in this example than the standard model. It catches both artificially planted anomalies in the dataset. While this is no indication of which one is better in a real-life noisy data setting, it is clear that we should test both in our further experiments, because of the presented results.

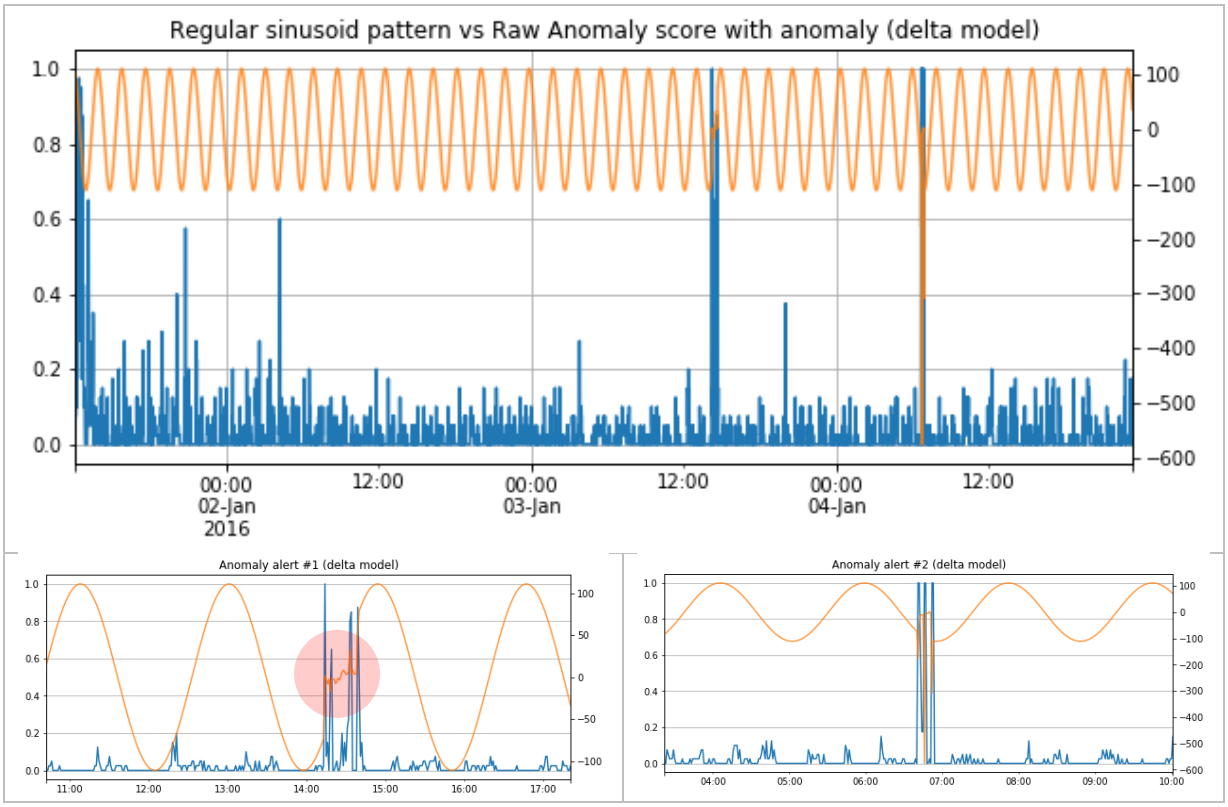

Fig. 5-6: The same sinusoid pattern with 2 artificially added anomalies as scored by the delta model. (just before 15:00pm Jan 3 and 07:00am Jan 04). The subplots show close-ups of the anomalies. Note: both anomalies are detected (with multiple hits) after achieving above 0.9 scores. The scale is visibly shifted, because of the relatively extremely large size of the delta value registered in the 2nd anomaly.

5.1.3 Anomaly Detection with HTM

In this series of experiments, the migraine sensor data was fed through the HTM anomaly detection model in a continuous (streaming) fashion, resembling a real life setting of online processing. The motivation was to discover if anomalies coincide with the adverse events, in this case migraine attacks. Since this is unsupervised learning, the target labels were used exclusively for the final evaluation of the result. The difference between the experiments consists of the type of running pre-processing methods used.

Data points (1 minute epochs for single-step, [2, …,15] minutes for aggregate) were entered to the model using the first 3 days (4320 data-points) for training. By "training" in this context we mean that the model learns the sequences, it receives no labels or feedback on what it is learning, i.e. unsupervised learning. As in the previous experiments, the model produced predictions (anomaly scores) for each timestep, which were ignored during the training period. After the training period, the anomaly score predictions were recorded for each timestep and appended into a separate result file. Anomaly scores with values over a specific threshold (*threshold = 1.0*) were counted as "hits", as described earlier in the sinusoid experiment.

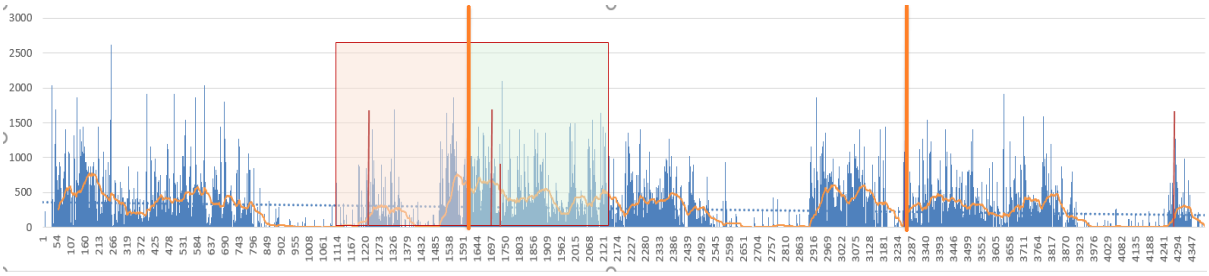

Fig. 5-7: An example of the segmentation of data into periods, centred around the target label timestamp. There are 2 hypothetical labelled attacks (thick orange verticals) in the figure. The 1st shows how the "hits" of anomaly predictions are segmented into the before (red) and after (blue) windows. Default size of each is P=120.

For the model to be evaluated, a period (P) before and after was marked for each consecutive target label as the target segments (Fig. 5-7). P was set to 120 timesteps, which translates into the period of P minutes. All hits counted within these segments were then summed into a score summary file together with anomaly likelihood and log of anomaly likelihood scores. An individual entry for each of the above scores in the result summary file was structured as follows:

[# hits in P before, # hits in P after, total # anomaly hits in total in current dataset]

An example extract from the score summary is illustrated in Listing 2:

```
#mp_h1_d0_w1_a05_n500_rad30_bst1_pam1_w51
2017-01-05 21:18,stm_aasane01_03_modified.csv,"[0, 0, 4]","[1, 3, 85]","[0, 0, 17]",5,3,23247
2017-01-05 21:27,stm_aasane16_modified.csv,"[0, 0, 9]","[0, 0, 48]","[0, 0, 11]",7,6,22150
2017-01-05 21:38,stm_aasane18_modified.csv,"[0, 0, 6]","[1, 2, 57]","[1, 1, 9]",4,4,25913
2017-01-05 22:01,stm_fusa04 (edited).csv,"[0, 0, 11]","[0, 0, 185]","[0, 0, 39]",0,0,47136
2017-01-05 22:14,stm_sb_modified.csv,"[0, 0, 9]","[2, 0, 77]","[1, 0, 21]",13,10,31488
#
```

Listing 2: A sample of score summary output file (early version) with 5-line batch. Color-coded scores: Raw Anomaly (red), Anomaly Likelihood (blue), Log Anomaly Likelihood (green). Format: [before, after, total]. Followed by total number of target labels, target labels outside training sequence, total data points. Prepended on each line the starting time and file name.

This was repeated with 6 to 7 complete datafiles, constituting a batch of the 4 positive sets, the rest being healthy controls[18] with varying model parameter configurations, and with the purpose to find the best parameters for optimal prediction accuracy. Over 200 such batches were produced with each taking an estimated 90-120 minutes to process. Each batch run was annotated with a header: the name of the model parameter file used to create the model. In Listing 2, it is '`mp_h1_d0_w1_a05_n500_rad30_bst1_pam1_w51`', meaning hours 'h1', weekends 'w1' are accounted for, 'd0', days of the week are not, 'a05', states alpha is 0.05, 'n500' the number of buckets, etc. A variety of model parameters were searched in the experiments: number of buckets, alpha (learning rate), boost values, Date encoder settings (weekends, time of day, day of week) and many more. The documentation details on these parameters were available

[18] In later experiments, the number of datasets in each batch was reduced to 4, due to time resource constraints.

through the Nupic source code (comments) on GitHub. Based on the score counts in the result summary file, the best parameters settings were discovered and reported for the final HTM model. As mentioned earlier, raw anomaly score and a related Anomaly likelihood score were kept for subsequent accuracy calculations based on the provided labels. Anomaly likelihood score is an alternative non-thresholding metric calculated in addition to the raw anomaly score. It is a probabilistic score that evaluates the current state to be anomalous based on a modelled distribution of historic anomaly scores from this model [59].

We started with single-timestep and scalar encoding, followed by aggregated scalar encoding and finally close with delta-encoding for single/aggregated step.

5.1.3.1 Single step, Scalar

In this setting single-step 1 minute epochs are processed. This is the same as the (effective) sample frequency of the sensor data. Scalar encoding is used for each timestep value and the anomaly predictions are therefore directly based on predicted vs actual value sequences. The top ten results are shown in Table 5-2.

Summary scores: non-aggregate scalar encoding							
stamp	**name**	**acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
08.02.2017 11:37	stm_aasane01_03	1	0,5	0,333	0,400	0,00013	FALSE
09.04.2017 14:50	stm_aasane16	0,9986	0,167	0,857	0,280	0,00033	FALSE
08.02.2017 11:19	stm_sb	0,9999	0,4	0,2	0,267	0,00033	FALSE
05.02.2017 10:13	stm_aasane01_03	0,9978	0,153	1	0,265	0,00013	FALSE
03.01.2017 21:37	stm_aasane01_03	0,9936	0,123	1	0,219	0,00013	FALSE
03.01.2017 21:47	stm_aasane16	0,9961	0,109	1	0,197	0,00028	FALSE
29.01.2017 20:48	stm_sb	0,9939	0,105	1	0,190	0,00033	FALSE
05.02.2017 19:32	stm_aasane01_03	0,9942	0,103	1	0,187	0,00013	FALSE
29.01.2017 15:43	stm_sb	0,9919	0,102	1	0,185	0,00033	FALSE
05.02.2017 11:13	stm_sb	0,9984	0,109	0,6	0,184	0,00033	FALSE

Summary scores: non-aggregate scalar encoding (new)							
stamp	**name**	**acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
10.04.2017 09:20	stm_aasane16	0,9991	0,308	0,5	0,381	0,00028	TRUE
10.04.2017 11:27	stm_aasane16	0,9991	0,308	0,5	0,381	0,00028	TRUE
11.04.2017 00:10	stm_aasane16	0,9991	0,308	0,5	0,381	0,00028	TRUE
11.04.2017 12:51	stm_aasane16	0,9991	0,308	0,5	0,381	0,00028	TRUE
06.05.2017 00:41	stm_aasane16	0,9985	0,162	0,5	0,245	0,00028	TRUE

06.05.2017 06:35	stm_aasane16	0,9985	0,162	0,5	0,245	0,00028	TRUE
07.05.2017 00:00	stm_sb	0,9968	0,139	0,8	0,237	0,00033	TRUE
07.05.2017 05:48	stm_sb	0,9968	0,139	0,8	0,237	0,00033	TRUE
10.04.2017 18:48	stm_sb	0,9941	0,138	0,8	0,235	0,00033	TRUE
11.04.2017 07:28	stm_sb	0,9941	0,138	0,8	0,235	0,00033	TRUE

Table 5-2 Top ranked results for non-aggregate (single-step) using scalar encoder and raw anomaly score as base for calculation. Entries are sorted by descending f1-score from precision/recall of positives. False value in "newRec" column indicates imperfect recall score calculations, from earlier runs. obtained by counting multiple hits in single target area.

5.1.3.2 Aggregate multi-step Scalar

Much like the previous experiment, except the data-points are now aggregated using the mean method (sum of N points/N) and this value is fed to the algorithm on every iteration step. This additional step performs simple smoothing of the data, decreasing the significance of outliers and noise on the overall result. Aggregation has been performed in step sizes of 2, 3 and 6 minute intervals. In more recent experiments ('newRecall' = True) the stride of the aggregation window can be adjusted independently. At the same time the number of individual data points processed by the model is reduced proportionally to the size of the aggregation stride. This has the additional effect of reducing the running time and it can potentially be an advantage in a real-life application, saving battery and network resources. The top results are shown in Table 5-3.

Summary scores: aggregate scalar encoding							
stamp	**Name**	**acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
05.01.2017 14:09	stm_sb	0,9994	0,452	1	0,623	0,00033	FALSE
05.01.2017 15:00	stm_sb	0,9994	0,452	1	0,623	0,00033	FALSE
07.02.2017 09:34	stm_aasane16	0,9994	0,435	1	0,606	0,00028	FALSE
05.02.2017 09:52	stm_aasane16	0,9998	0,5	0,667	0,572	0,00028	FALSE
07.02.2017 11:50	stm_aasane16	0,9994	0,4	1	0,571	0,00028	FALSE
05.01.2017 11:40	stm_sb	0,999	0,375	1	0,545	0,00033	FALSE
29.01.2017 12:49	stm_sb	0,9973	0,362	1	0,532	0,00033	FALSE
12.01.2017 09:42	stm_aasane01_03	0,9909	0,353	1	0,522	0,00013	FALSE
12.01.2017 09:51	stm_sb	0,9899	0,351	1	0,520	0,00033	FALSE
05.01.2017 14:27	stm_sb	0,9992	0,342	1	0,510	0,00033	FALSE
Summary scores: aggregate scalar encoding (new)							
stamp	**Name**	**acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
07.05.2017 21:23	stm_aasane16	1	1	0,25	0,400	0,00063	TRUE
08.05.2017 00:13	stm_aasane16	1	1	0,25	0,400	0,00063	TRUE

07.05.2017 23:36	stm_aasane16	0,9998	0,5	0,25	0,333	0,00063	TRUE
08.05.2017 02:25	stm_aasane16	0,9998	0,5	0,25	0,333	0,00063	TRUE
09.05.2017 02:05	stm_aasane01_03	0,9914	0,183	0,667	0,287	0,00044	TRUE
09.05.2017 07:36	stm_aasane01_03	0,9914	0,183	0,667	0,287	0,00044	TRUE
07.05.2017 21:32	stm_aasane16	0,9997	0,333	0,25	0,286	0,00063	TRUE
08.05.2017 00:22	stm_aasane16	0,9997	0,333	0,25	0,286	0,00063	TRUE
10.05.2017 00:30	stm_aasane16	0,9903	0,162	0,667	0,261	0,00094	TRUE
10.05.2017 04:16	stm_aasane16	0,9903	0,162	0,667	0,261	0,00094	TRUE

Table 5-3: Top ranking sorted summary scores (sorted by f1-score) from aggregate scalar encoder experiments. The "newRec" column (all false values) indicate imperfect Recall score calculations, obtained by counting multiple hits in single target area.

5.1.3.3 Delta encoding

Delta encoding [38], is an alternative to encoding sequences of individual data points, where the sequences of absolute change values is memorized and subsequently predicted by the model. In an anomaly detection setting, this will flag the rarely seen sequences, and completely ignore the actual sensor values.

The idea of this experiment is the assumption that certain activity patterns might be specifically distinct relative to the power of preceding sensor readings and not depend only on the actual value. This basically means that we attempt to find patterns in a chain of pairs of activity value readings. In practice, the input to the model again consists of single step 1 min epochs, while the output score is now related to the expected difference in values (i.e. the delta value).

Summary scores: non-aggregate delta (raw)							
stamp	**name**	**Acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
03.05.2017 21:58	stm_aasane16	0,9985	0,162	0,5	0,245	0,00028	TRUE
04.05.2017 03:51	stm_aasane16	0,9985	0,162	0,5	0,245	0,00028	TRUE
05.05.2017 00:35	stm_aasane16	0,9985	0,162	0,5	0,245	0,00028	TRUE
05.05.2017 06:30	stm_aasane16	0,9985	0,162	0,5	0,245	0,00028	TRUE
01.05.2017 23:19	stm_aasane01_03	0,9998	0,167	0,333	0,222	0,00013	TRUE
02.05.2017 10:32	stm_aasane01_03	0,9998	0,167	0,333	0,222	0,00013	TRUE
02.05.2017 00:02	stm_sb	0,9991	0,161	0,3	0,210	0,00033	TRUE
02.05.2017 11:14	stm_sb	0,9991	0,161	0,3	0,210	0,00033	TRUE
02.05.2017 06:34	stm_sb	0,9953	0,118	0,8	0,206	0,00033	TRUE
30.04.2017 23:12	stm_sb	0,988	0,114	1	0,205	0,00033	TRUE

Summary scores: non-aggregate delta (likelihood)							
stamp	**name**	**Acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
03.05.2017 02:58	stm_aasane16	0,9994	0,52	0,5	0,510	0,00028	TRUE
04.05.2017 00:23	stm_aasane16	0,9991	0,394	0,5	0,441	0,00028	TRUE
05.05.2017 08:55	stm_aasane16	0,9991	0,394	0,5	0,441	0,00028	TRUE
30.04.2017 06:27	stm_aasane16	0,9994	0,381	0,5	0,432	0,00028	TRUE
02.05.2017 23:58	stm_aasane16	0,9988	0,35	0,5	0,412	0,00028	TRUE
03.05.2017 00:29	stm_aasane16	0,9986	0,333	0,5	0,400	0,00028	TRUE
03.05.2017 06:39	stm_aasane16	0,9986	0,333	0,5	0,400	0,00028	TRUE
01.05.2017 00:49	stm_aasane01_03	0,9982	0,397	0,333	0,362	0,00013	TRUE
01.05.2017 05:11	stm_aasane01_03	0,9982	0,397	0,333	0,362	0,00013	TRUE
30.04.2017 02:20	stm_aasane01_03	0,9978	0,377	0,333	0,354	0,00013	TRUE

Table 5-4: Top ranked results from non-aggregate delta encoded experiments. Top: results obtained from raw anomaly scores. Bottom: results based on anomaly likelihood scores.

Summary scores: aggregate delta (raw)							
stamp	**name**	**acc**	**prec**	**recall**	**F1-score**	**preval**	**newRecall**
11.05.2017 22:49	stm_aasane16	0,9951	0,184	0,667	0,288	0,00094	TRUE
12.05.2017 02:39	stm_aasane16	0,9951	0,184	0,667	0,288	0,00094	TRUE
11.05.2017 22:40	stm_aasane16	0,995	0,179	0,667	0,282	0,00094	TRUE
12.05.2017 02:30	stm_aasane16	0,995	0,179	0,667	0,282	0,00094	TRUE
11.05.2017 23:35	stm_aasane16	0,9928	0,164	0,667	0,263	0,00094	TRUE
12.05.2017 03:25	stm_aasane16	0,9928	0,164	0,667	0,263	0,00094	TRUE
11.05.2017 23:02	stm_sb	0,9904	0,142	0,636	0,232	0,00116	TRUE
12.05.2017 02:52	stm_sb	0,9904	0,142	0,636	0,232	0,00116	TRUE
12.05.2017 08:17	stm_sb	0,9869	0,133	0,636	0,220	0,00116	TRUE
12.05.2017 09:05	stm_sb	0,9869	0,133	0,636	0,220	0,00116	TRUE

Table 5-5: Top ranked results from aggregate (size 6, stride 3) delta encoder experiments. Scores are based on the raw anomaly scores obtained.

The results shown above are the truncated top-ranking models in each category. The complete score summary files are supplied with the archive.

The best consistent results overall are obtained when using anomaly likelihood score rather than raw anomaly scores in combination with delta encoding (see Table 5-4 for single-step and Table 5-5 for aggregated results). We should add that we used much less time running delta experiments (due to limited time and resource constraints) and believe that with more extensive parameter tuning, better results may be possible.

The best performing models, although showing relatively poor results (F1-score ≤ 0.5) do in fact seem to support the findings from [70], where skewness and mean showed statistically significant differences from before a migraine attack to after the start timestamp of the attack (termed as the "during" period in the paper). The ratio of hits in the before window to the after window is 5:14 in favour of the after window (see Listing 3). This is not what we would normally expect from a distribution where all windows have equal hit probabilities, i.e. random. Similar observation can be made in other well performing models (see supplied summary files).

```
#
# mp_h1_d0_w1_a05_n400_rad30_bst1_pam1 (MEAN step=5)
2017-01-05 14:09, stm_sb_modified.csv,"[5, 14, 31]","[0, 0, 0]","[0, 0, 0]",13,10,31488
```

Listing 3: The entry corresponding to the top scorer from summary file in Table 4-1. Notice the 5:14 ratio of hits in the before/after windows. This ratio has consistently turned out to the advantage of 'after' window, potentially indicating a significant difference from the rest.

It must be added that some of the early results suffer from incompleteness issues, due to the way early counting of hits was set up. The calculation was based on total number of hits in the target areas in relation to the count of target areas, but multiple hits inside the same target area were counted equally to those with single hits. Because of this, the Recall score obtained from the early versions cannot be considered correct. This is fixed in later runs, by extending the result with a 3-tuple of correctly calculated Recall scores. Due to the long processing time of each file (10-15 mins per individual dataset), we did not repeat the almost 500 earlier runs. The corrected results are annotated accordingly in the results table using the "newRec" column (True annotations depict results with correctly calculated recall).

5.2 Classification experiments

Our motivation for using classification for this problem stems from the fact that based on the studied literature, such problems are traditionally solved, applying similar methods. The closest analogy is mood detection and ADL detection as described in earlier chapters. We have labelled data, two distinct classes, i.e. a binary classification problem of attack vs no attack. Given that, we train classifiers on a subset of the data and evaluate performance on a hold-out set. Furthermore, we expect to be able to compare the two systemic approaches as part of the analysis.

It has been shown that there are significant variations in activity patterns, as recorded by an wearable actigraph, between healthy and diagnosed subjects, as well as in periods before, during and after the migraine attacks in patients [70]. These variations should consequently be detectable in the patterns formed by the underlying sequence data. Through these experiments, we attempt to build the optimal classification model to recognize attack periods with highest possible accuracy. As a secondary objective, we attempt to derive the best performing combination of preprocessing techniques, classifiers and hyper-parameters.

The result tables presented in this section are coded with abbreviations of metrics and experiment parameter settings. An explanation of these codes is found in 8Appendix C .

5.2.1 Basic feature set

In this classification experiment we create a base set of 20 statistical features for every window created with the sliding window process. Every such window now represents an instance, consisting of feature attributes and a designated target label. It is important to mention that the statistics are calculated independently for each instance, i.e. no information is made available on the statistical variables of the parent dataset. Features calculated for this purpose include: Mean, standard deviation, skewness, kurtosis, quantile (0.25), quantile (0.75), quantile (0.90), quantile (0.15), median, mean absolute deviation, standard error of the mean (SEM), autocorrelation ACF (lag 1-5), mode (most frequent value). Notably, we have made an attempt to include the highest scoring features as described in [43] among them. The complete list is presented in Table 5-6, below.

1.	Arithmetic mean	$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
2.	Standard deviation	$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$
3.	Skewness	$S = \left[\frac{n}{(n-1)(n-2)}\right]\sum_{i=1}^{n}\frac{(x_i - \bar{x})^3}{s^3}$
4.	Kurtosis (excess)	$k = \frac{n\sum_{i=1}^{n}(x_i - \bar{x})^4}{\left(\sum_{i=1}^{n}(x_i - \bar{x})^2\right)^2} - 3$
5.	Quantile (15%)	$Q_{15} = \frac{15}{100}(n+1)$
6.	Quantile (25%)	$Q_{25} = \frac{25}{100}(n+1)$
7.	Quantile (75%)	$Q_{75} = \frac{75}{100}(n+1)$
8.	Quantile (90%)	$Q_{90} = \frac{90}{100}(n+1)$
9.	Median (middle item in sorted list)	$M = M_{(n+1)/2}$ $M = \frac{M_{\frac{n+1}{2}} + M_{\frac{n+1}{2}+1}}{2}$
10.	MAD, Mean Absolute Deviation average distance between each data value and the mean	$MAD = \frac{1}{n}\sum_{i=1}^{n}\lvert x_i - \bar{x}\rvert$
11.	SEM, Standard error of the mean in a sample	$s_m = \frac{s}{\sqrt{N}}$
12.	Unbiased sample variance	$s^2 = \frac{\sum(X-M)^2}{N-1}$
13.	Autocorrelation (lag 1)	$\hat{\rho}_k = \frac{\sum_{t=k+1}^{T}(r_t - \bar{r})(r_{t-k} - \bar{r})}{\sum_{t=1}^{T}(r_t - \bar{r})^2}$
14.	Autocorrelation (lag 2)	
15.	Autocorrelation (lag 3)	
16.	Autocorrelation (lag 4)	
17.	Autocorrelation (lag 5)	
18.	Mode (most frequently occurring value)	$M_{f_{max}}$

Table 5-6: Statistical features that were calculated for each individual instance used in classification experiments.

5.2.2 Early (flawed) results: mixed window size, SOS

Early classification experiments were executed with a limited base feature set consisting of 16 features: feature no.10 (MAD) and no.18 (Mode) were left out (see Table 5-6). The early experiment runs were executed with mixed stride values and window sizes. Additionally, our simplified oversampling (SOS) method was used.

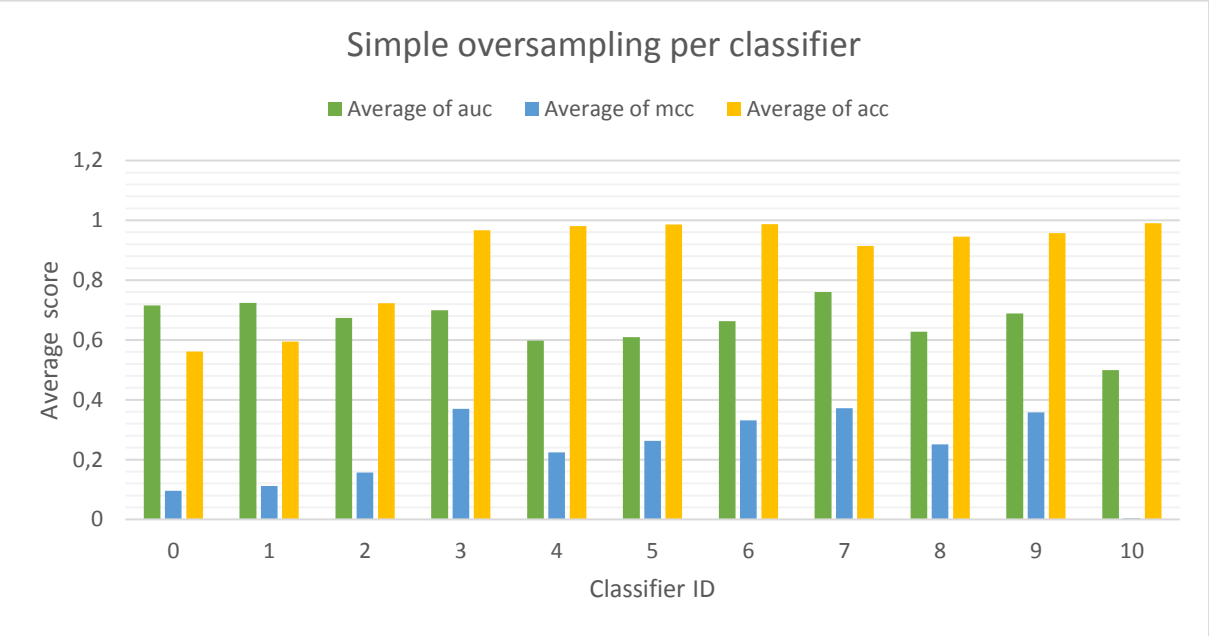

Fig. 5-9: Average scores per classifier in the preliminary experiments. A notable point is the relative difference in ACC scores between id.7 and id.4, as id.7 performs better even with lower ACC score as discussed in section 3.2.4.2 on metrics.

test	alg	acc	mcc	kap	auc	geo	iba	TN	FP	FN	TP	pr	re	f1	feats
1	7	0,989	0,834	0,828	0,967	-1	-1	1808	17	3	50	0,99	0,99	0,99	16
5	7	0,992	0,823	0,818	0,958	-1	-1	3054	21	5	60	0,99	0,99	0,99	16
4	9	0,995	0,887	0,886	0,953	-1	-1	3065	9	6	60	1	1	1	16
4	7	0,993	0,844	0,842	0,952	-1	-1	3058	16	6	60	0,99	0,99	0,99	16
4	3	0,996	0,892	0,892	0,946	-1	-1	3067	7	7	59	1	1	1	16
6	7	0,993	0,843	0,841	0,945	-1	-1	3058	15	7	60	0,99	0,99	0,99	16
5	9	0,996	0,897	0,897	0,945	-1	-1	3069	6	7	58	1	1	1	16
3	7	0,994	0,861	0,861	0,944	-1	-1	3065	11	7	57	0,99	0,99	0,99	16
2	7	0,982	0,758	0,748	0,941	-1	-1	1791	28	6	53	0,99	0,98	0,98	16
4	6	0,995	0,874	0,874	0,931	-1	-1	3067	7	9	57	0,99	0,99	0,99	16

Fig. 5-10: Top 10 scores for the mixed/SOS early experiments sorted by AUC. These results were obtained using an early version of the basic feature set using our simple oversampling technique, based on adjacent/overlapping sequences. ('test': batch id, 'alg': id of algorithm used, 'feats': feature count). Compare to Fig. 5-9.

The disadvantage of the SOS method became evident when applying the random train-test set split (as well as later during the cross-validation loop). At first nothing seemed to be out of the

ordinary, with one exception: the results were surprisingly good, especially if we consider the findings from the exploratory data analysis. It was when the SMOTE method was used and a comparison of results made that serious suspicion arose.

On closer inspection, it was discovered that this anomaly was caused by the process of splitting the data. Synthetic instances, which we know to be strongly similar, were sampled randomly from the data for the training set in the same way as the originals. However, the originals (as mentioned in exploration section) do not actually show these high levels of similarity. Since SMOTE synthesizes the final instance (feature vector created for some underlying sequence) and not the underlying sequence itself, the final effect is two very different instances. This could also be the reason behind SMOTE experiments' weak positive effect on classifier performance.

A second flaw, probably more serious was that some synthetic instances created based on the complete data set, were then leaked into the test set. Both have been corrected in the remaining experiments.

5.2.3 Extended feature set

In order to investigate the importance of a large amount of complex features we decided to greatly extend the basic feature set. Increasing the number of features often, but not always leads to improved classification performance and in fact, often may lead to worse results [93]. The inclusion of more complex features (as compared to the basic set), such as spectral features, frequency, entropy and wavelet functions should also broaden the discriminative power of the model. Incidentally, the approximate entropy measure of regularity is contained within this set, a significant feature to include according to [12]. As the task of correctly implementing a huge number of complex features is outside the scope of this work, not to mention time consuming, we decided to search for 3rd party libraries providing this functionality. The "TSfresh" library [94] serves this purpose, though newer versions have caused instability and system crashes. For this reason, a slightly older version 0.5 is used.

The 67 base methods listed in 8Appendix C are used to derive the features, some requiring further parameters and some producing multiple outputs. A total of 217 individual, extended features are extracted with no modifications done to the default parameters. The actual number extracted is higher but feature columns with NaN-values (Not a Number) were removed prior to further processing. The set of extended features is never combined with the basic set; however, we note the following overlaps: autocorrelation, kurtosis, mean, median, quantile, skewness, standard deviation and variance.

Fig. 5-11: Average scores for experiments (200x25) using the extended feature set. Classifiers are sorted from highest MCC score.

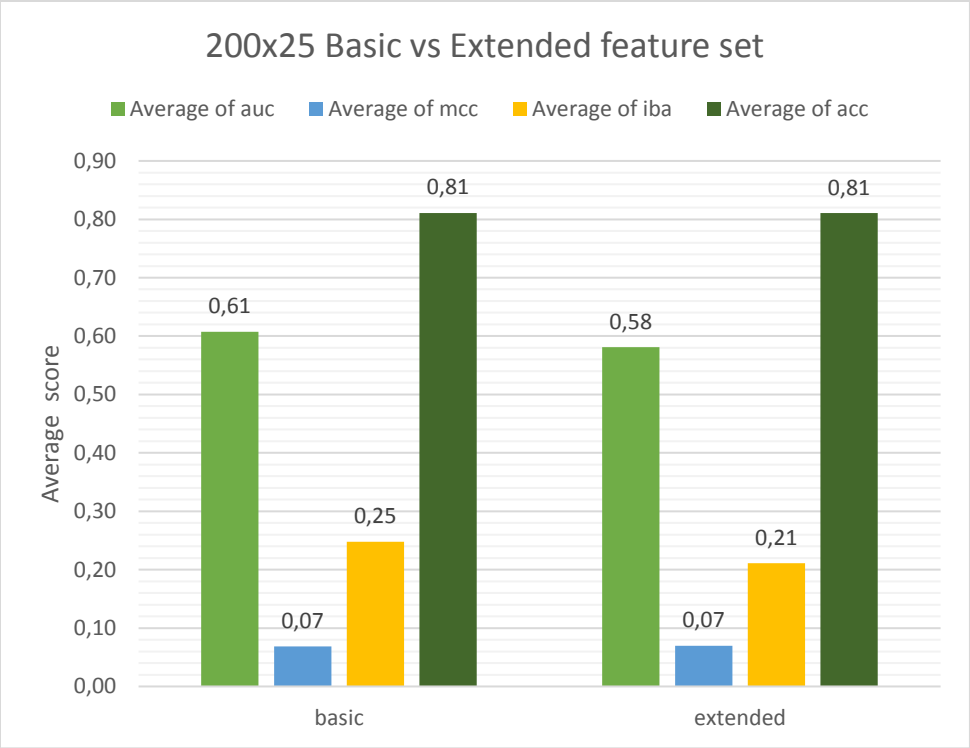

Fig. 5-12: Average scores obtained for each type of feature set (basic vs extended).

Fig. 5-11 represents the total average scores (AUC, MCC, IBA) obtained exclusively from experiment runs using the extended feature set. Fig. 5-12 shows a direct comparison of total average scores (including ACC) between the basic and extended feature sets. We observe a slight advantage of basic over extended (AUC & IBA). It is not however, reflected in MCC or ACC scores.

5.2.4 Additional features

In addition to the selected feature set (basic or extended) described above, most experiments were executed using combinations of the specially created domain features. As described in earlier sections (4.3.3) these include skewness delta, kurtosis delta, quality of sleep and time of day. The lags for the kurtosis/skewness feature are chosen from the set {1, 2, 5}. The default value is L=1, meaning that L previous windows' skewness/kurtosis values are added to the current instance features.

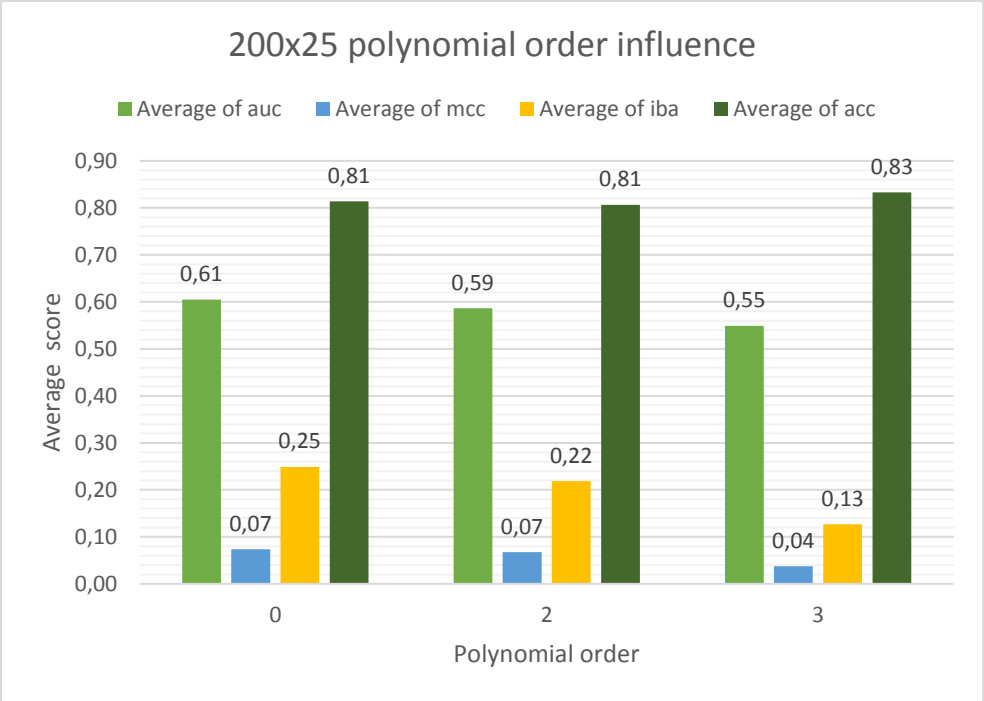

Fig. 5-13: Influence of polynomial feature extraction on average score of the mixed 200x25 experiments.

As the final step of feature extraction, polynomial features are optionally created from the total set of features, further extending the set. The order of the polynomial used is considered depending on the size of the set prior to avoid creating an explosion in number of features. For the basic set, we choose from the set {0, 2, 3}, while for the much bigger extended set, we limit the max allowed order value to 2. Fig. 5-13 summarizes the overall effect of polynomial features on average scores. The impact does not seem to be huge, but it must be noted that some classifiers like SVC with non-linear kernels may not be impacted by this.

5.2.5 Filters, PCA, K-select

As a further measure, two different smoothing filters, Tukey [95] and Hodrick-Prescott (HP) filter were applied individually and in combinations. Tukey belongs to the group of windowing functions for smoothing values, also known as tapering functions. These filters are generally used for performing apodization[19], i.e. smoothing of discontinuities from the outer edges of the window. The HP-filter is used mainly in financial domains for detrending and smoothing of time series [96]. The influence of filtering on performance is shown in Fig. 5-14 and Fig. 5-15.

[19] Meaning literally «removal of the foot».

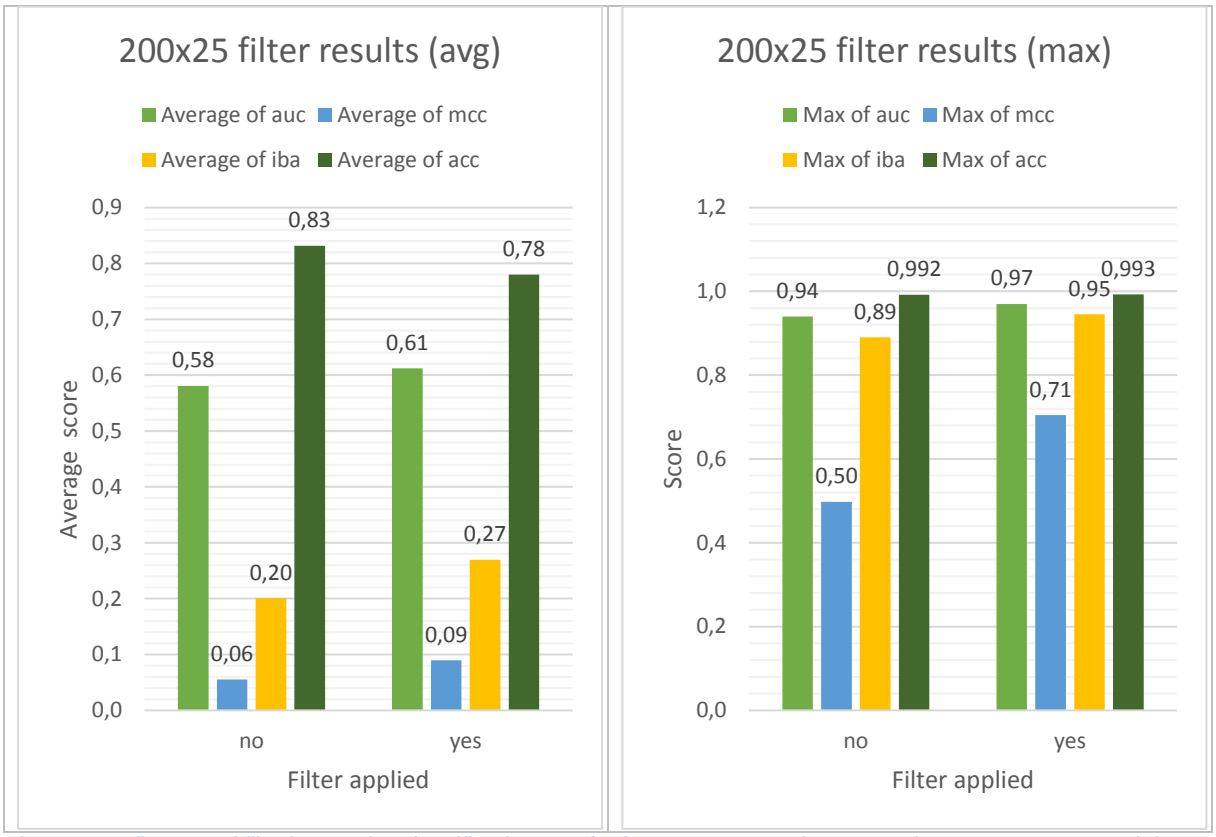

Fig. 5-14: Influence of filtering on the classification results for 200x25 experiments. Left: Average scores, Right: Maximum scores obtained.

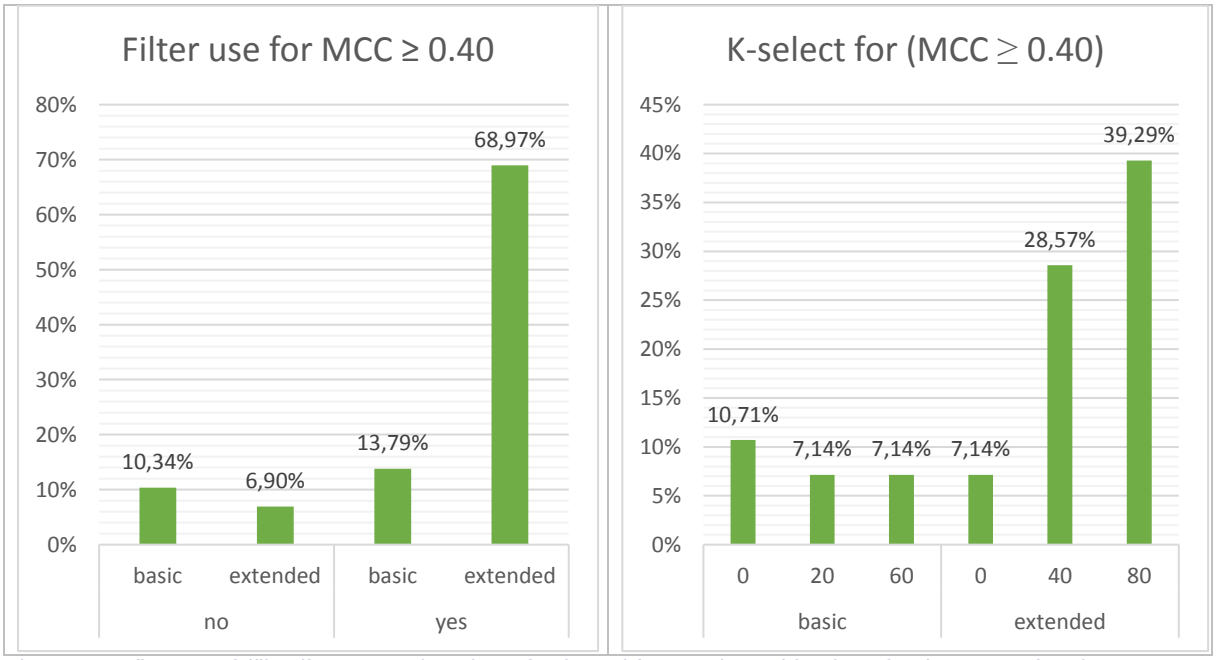

Fig. 5-15: Influence of (filter/feature set/number of selected features) combinations for the top performing classifiers (according to MCC). Left: Filtering + feature set type. Right: Feature selection (K-select) + feature set type. Values are in percent of top scorers with MCC ≥ 0.40.

Selected experiment runs are processed with dimensionality reduction techniques. Principal Component Analysis (PCA) and a simple feature selection algorithm, K-select [20] are optionally applied reducing the number of features to *k* most important ones. An important factor to consider is the number of features before applying these techniques. We have set the set of possible values to $k \in \{20, 40, 50, 60, 80, 100, 200\}$. The number of features before processing must be at least *k*, so most combinations require either the extended feature set or a basic set extended by polynomial features. K-select results on the top scoring experiments is shown in Fig. 5-15 (right side).

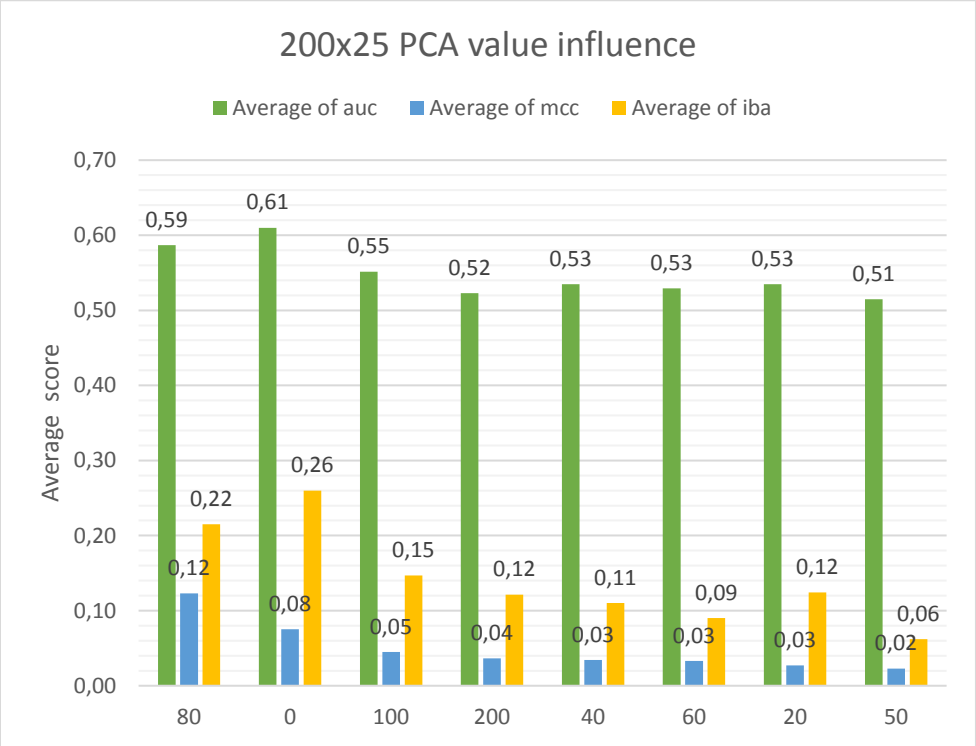

Fig. 5-16: Influence of PCA on the average results for 200x25 experiments, sorted by MCC. (Note: applying a value of x implies that the total number of features before processing is ≥ x).

All features are standardized (with 0 mean), as this is a requirement for the correct functionality [24] of the classification algorithms used. PCA influence on average performance of the classifiers is presented in Fig. 5-16. The minimum feature count requirement described for K-select applies also to PCA, i.e. in practice PCA values of over 20 can only be used on the extended dataset (or basic dataset expanded with polynomial features).

5.2.6 Class balancing

Class balancing methods (see section 3.2.5) were optionally applied in a number of experiments in order to create grounds for comparison of their effectiveness in combination with the other methods used. We have selected two best performing methods according to the mini-experiment described in section 4.1, i.e. SMOTE and CC.

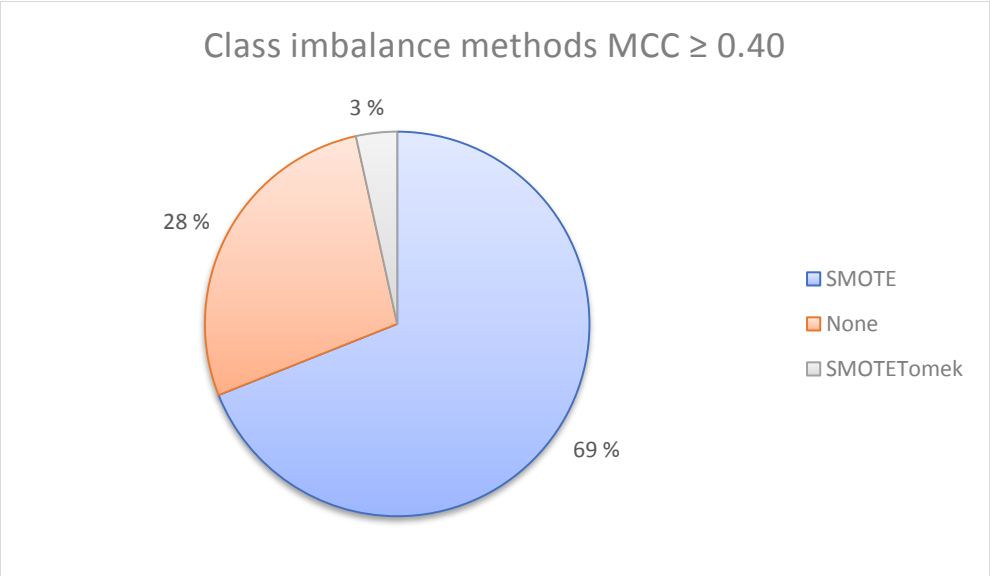

Fig. 5-17: Class balancing method impact in top ranking results using MCC measure.

These were used in the majority of experiments where class balancing was employed. Additionally, we applied a few variants of these in selected experiment runs (where execution times were expected to be relatively short, not impacting the process heavily). One of these variants (SMOTETomek) has managed to obtain scores good enough to end up in the top results. The distribution of these methods among top ranking results using MCC and AUC, respectively are presented below (Fig. 5-17 and Fig. 5-18). We observe very strong domination of oversampling methods, specifically SMOTE, while under-sampling (CC) shows mixed results.

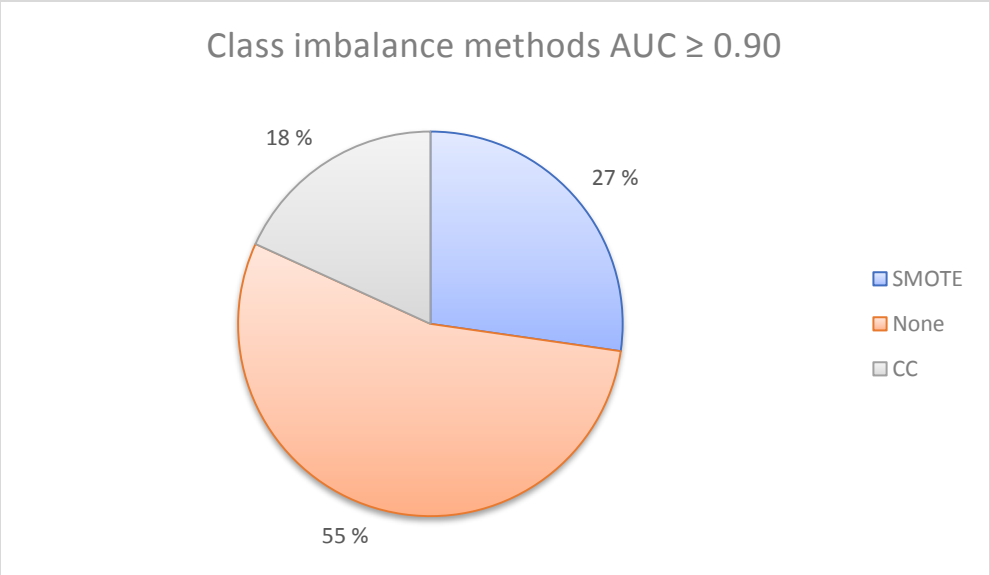

Fig. 5-18: Class balancing method impact in top ranking results using the AUC metric.

5.2.7 Final results

The best results were obtained using a combination of methods and features described above. The results are based on classifiers optimized to maximize the MCC-score, using 5-fold cross validation technique. We managed to obtain a MCC score of 0.705 (AUC 0.75) with 0 FP values, 2 TP and 2 FN out of a total of 4 positives (see table Fig. 5-19: Top 10 results 200x25 sorted by MCC-score. and confusion matrix in Fig. 5-20). This is the result that minimizes the number of false positives and therefore avoiding potential false alarms.

alg	acc	mcc	kap	auc	geo	iba	TN	FP	FN	TP	pca	poly	ksel	exs	feats	imb	filter	set
8	0,993	0,705	0,664	0,75	0,707	0,475	296	0	2	2	0	0	80	998	220	1	yes	extend
8	0,99	0,572	0,566	0,748	0,706	0,474	295	1	2	2	80	2	80	998	25425	1	yes	extend
6	0,99	0,572	0,566	0,748	0,706	0,474	295	1	2	2	0	0	40	998	222	1	yes	extend
3	0,99	0,572	0,566	0,748	0,706	0,474	295	1	2	2	80	2	40	998	25425	0	yes	extend
6	0,992	0,498	0,397	0,625	0,5	0,231	372	0	3	1	60	3	60	1252	2024	0	no	basic
6	0,992	0,498	0,397	0,625	0,5	0,231	372	0	3	1	0	0	200	1252	438	1	no	extend
8	0,99	0,497	0,397	0,625	0,5	0,231	296	0	3	1	80	2	80	998	24976	0	yes	extend
8	0,99	0,497	0,397	0,625	0,5	0,231	296	0	3	1	80	2	80	998	25425	0	yes	extend
5	0,99	0,497	0,397	0,625	0,5	0,231	296	0	3	1	0	0	40	998	222	0	yes	extend
5	0,99	0,497	0,397	0,625	0,5	0,231	296	0	3	1	0	0	80	998	220	0	yes	extend

Fig. 5-19: Top 10 results 200x25 sorted by MCC-score.

Best by MCC-score

	Predicted Condition	
Actual Condition	Normal	Attack
Normal	296	0
Attack	2	2

Best by AUC-score

	Predicted Condition	
Actual Condition	Normal	Attack
Normal	278	18
Attack	0	4

Fig. 5-20: Confusion matrix for the best scoring models. Left: the best scoring (by MCC-score) id.8 (Voting classifier) with score of 0.705 and AUC 0.75. Right: Best scoring (by AUC-score), id.9 (KNN classifier) with AUC of 0.97 and MCC of 0.413.

The second, equally promising result is the best classifier with AUC-score of 0.97 (MCC 0.413), with confusion matrix presented in Fig. 5-20. The detailed top 10 performers sorted by AUC-score are presented in Fig. 5-21. In this case, the model correctly finds all attacks, but also misclassifies 18 negatives as attacks. It is somewhat surprising that KNN using a basic feature set ends up on the 2 top spots. The dominant classifier in both top scoring charts is id.3, Extra-Trees.

alg	acc	mcc	kap	auc	geo	iba	TN	FP	FN	TP	pca	poly	ksel	exs	feats	imb	filter	set
9	0,94	0,413	0,292	0,97	0,969	0,945	278	18	0	4	0	0	20	998	22	1	yes	basic
9	0,923	0,37	0,24	0,961	0,96	0,929	273	23	0	4	0	0	0	998	38	1	yes	basic
3	0,918	0,324	0,19	0,958	0,957	0,924	341	31	0	4	0	0	0	1252	22	0	yes	basic
3	0,88	0,268	0,134	0,94	0,938	0,89	327	45	0	4	0	0	40	1252	222	0	no	extend
3	0,878	0,265	0,131	0,938	0,936	0,887	326	46	0	4	0	2	60	1252	780	0	no	basic
3	0,867	0,253	0,121	0,933	0,93	0,877	322	50	0	4	0	2	0	1252	276	0	no	basic
2	0,84	0,229	0,1	0,919	0,916	0,852	312	60	0	4	0	0	20	1252	22	1	yes	basic
3	0,83	0,221	0,093	0,914	0,91	0,842	308	64	0	4	60	2	60	1252	741	0	no	basic
3	0,816	0,211	0,085	0,907	0,903	0,83	303	69	0	4	0	0	0	1252	22	0	no	basic
5	0,811	0,208	0,083	0,905	0,9	0,825	301	71	0	4	0	2	40	1252	24976	2	no	extend

Fig. 5-21: Top 10 results 200x25, sorted by AUC-score.

To obtain a more accurate overview over which classifiers showed the best performance overall, we created additional queries. First, out of all classifiers scoring MCC 0.40 or better we select the best obtained MCC scores for each (Fig. 5-22). Apparently, 4 out of 10 did not once manage to obtain a required minimum score (MCC ≥ 0.40) and do not appear in this chart (id: 0, 1, 4, 7).

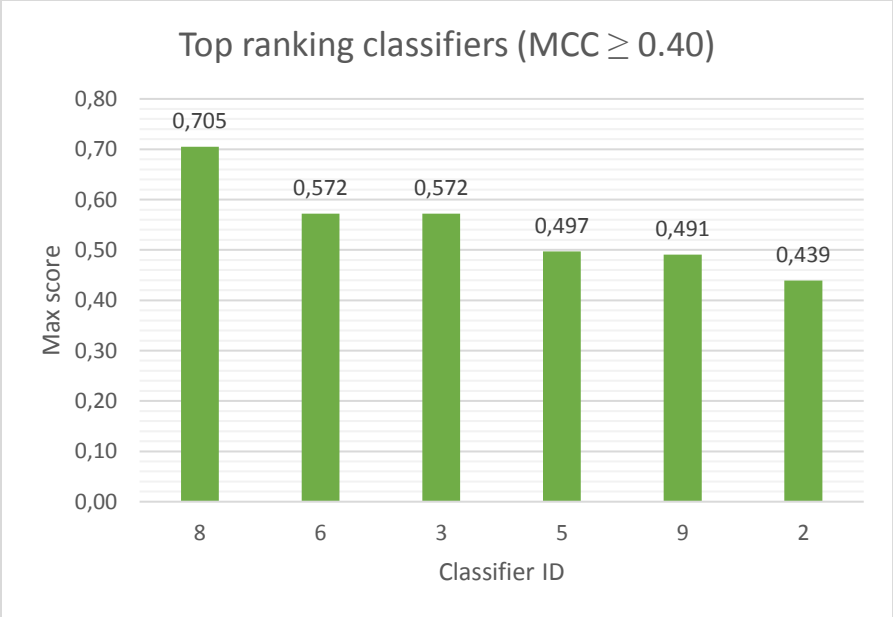

Fig. 5-22: Best scored classifiers with the requirement that MCC ≥ 0.40.

Fig. 5-23 presents a more specific picture of the frequency of each classifier in the top scoring group of entries with scores MCC ≥ 0.40 and AUC ≥ 0.90 respectively. We can clearly see that classifier id.3 (Extra-Trees) dominates the best results evaluated using AUC, while id.5 and 8 (MLP and Voting) amount for over 55% of the best results measured with MCC.

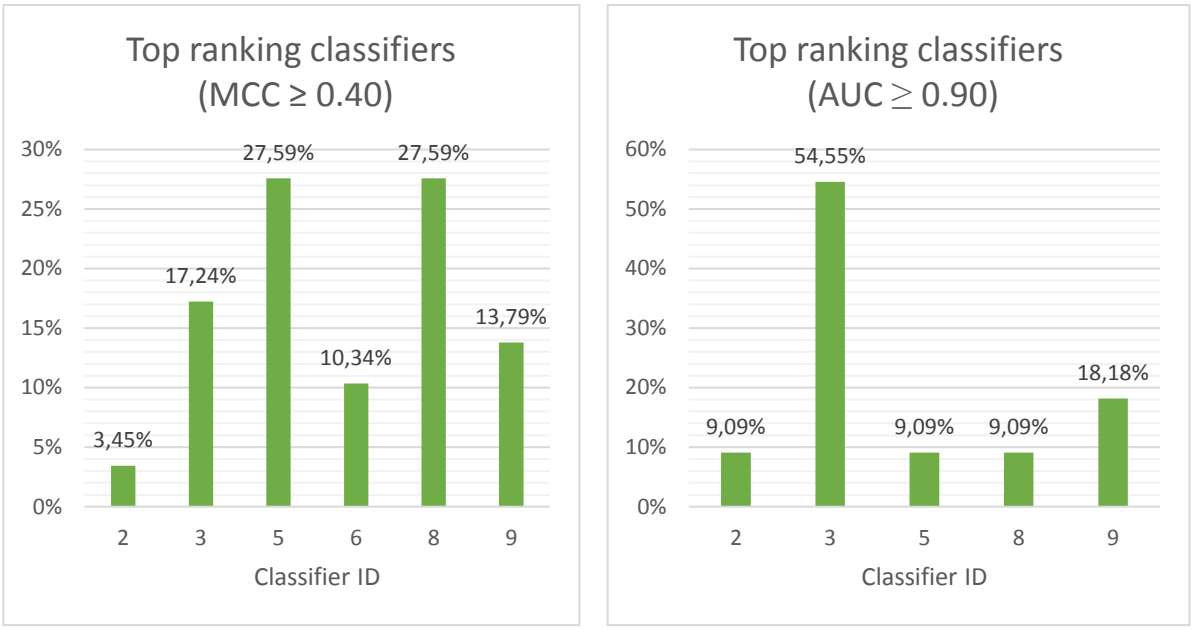

Fig. 5-23: Percentage of classifiers with top scoring entries in respective ranges (MCC ≥ 0.40 and AUC ≥ 0.90).

Finally, we should point out an additional factor that could be of interest, namely the column named "*exs*" in the result tables. This column specifies the number of examples in the train set for that experiment, but what is more important, it corresponds directly to whether the data was cleared of inactive periods (an artefact of the actigraph watch being taken off for longer periods) or not. Upon a closer look, we observe a dominance of the cleaned datasets, i.e. the '998' vs '1252' (see Fig. 5-19 and Fig. 5-21). Fig. 5-24 shows the result of comparing the average scores obtained for each group.

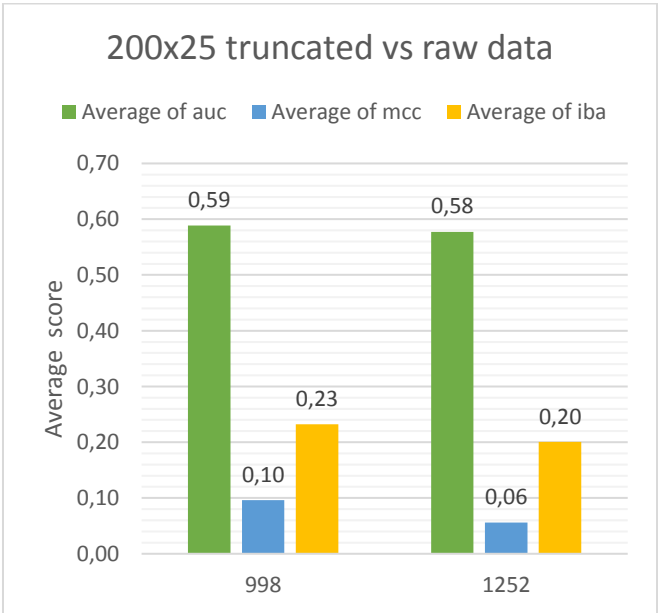

Fig. 5-24: Average scores obtained for data with artefacts removed (label 998), and for data with artefacts (1252).

All the above results focus on one specific sliding window setting: window size of 200 and stride of 25. A total of 2760 experiments have been recorded using these settings. Naturally, we have explored other window settings (early result logs and summaries are included separately), but due to time constraints we were forced to settle on one. Windows sizes of {100, 200, 500, 1000, 2000} and strides of {2, 3, 10, 25, 30, 44} were briefly tested in preliminary experiments. The majority of migraine attacks lasted on average 2-3 hours [70]. Choosing 200 timesteps was partly due to the need to minimize the window size to provide a more realistic scenario. 200 timesteps corresponds to 3 hours and 20 minutes, while a window of 1000 is over 16 hours. The second reason was the initial poor results from other window sizes, e.g. 500x25 as shown in Fig. 5-25. It must be said that preliminary experiments only tried a minimal number of combinations.

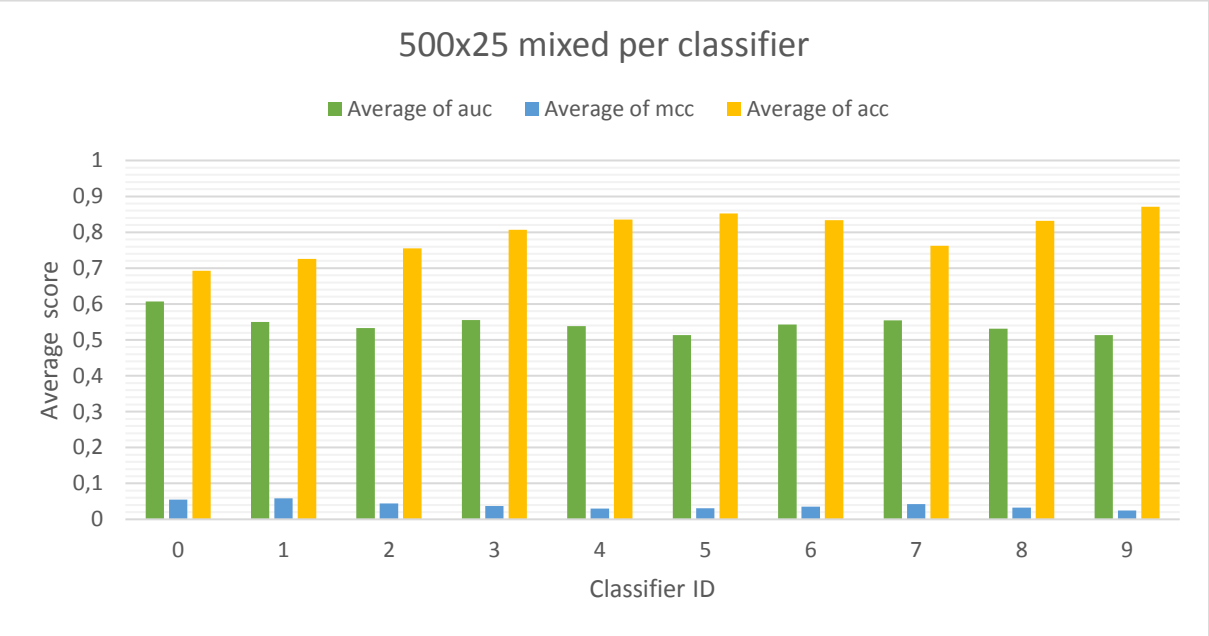

Fig. 5-25: Average results all experiments over 500x25 (size=500, stride=25). SOS was not applied.

6 Discussion

We set out with several objectives for this work. The first was to determine the feasibility of applying supervised machine learning techniques to the detection of patterns related to periods of transition states prior to bipolar mood episodes. In the absence of specific bipolar dataset, we settled for a migraine attacks dataset. The objective changed to detection of patterns related to migraine attacks. In context of RQ1, we found that analysis of given actigraph signals using machine learning has some potential for detection of temporal activity patterns experienced during migraine attacks. We base this on results obtained from a variety of experiments using supervised (section 3.2.1) and unsupervised (section 3.2.1.8) methods.

Through a number of experiments, we found specific algorithms and feature combinations which result in better overall classification performance with this type of dataset. The top result obtained an AUC score of 0.97 and MCC score of 0.41, with a KNN classifier. This result included a number of false positives (FP), but classified all attacks correctly. FP above zero may not be desirable in a system that must potentially provide a warning in case of positive detection. For this reason, we additionally trained classifiers to optimize the MCC score. The winner, the Voting classifier obtained AUC of 0.75 and MCC of 0.71, with zero FP count. This in turn, provided the best result with zero FP count, but at the cost of some (50%) false negatives (FN). FN count above zero in practice means some attacks were not detected, however in this scenario the warning could potentially be given not risking false alarms.

On the whole, we obtained promising results from several classification experiments. Although Extra-Trees (id.3) did not end up as the best individual winner of any experiment, it has shown an overall good performance dominating among the top-ranking classifiers. KNN (id.9) and Voting ensemble (id.8) consisting of combined Logistic Regression, Random Forest and KNN have multiple times provided the best results. An interesting proposition would be to substitute the Extra-Trees element for Random Forest in the ensemble, something we leave for future attempts. A more intriguing fact is KNN obtaining the best result, even after we found evidence that the underlying sequences will not respond to classification using distance measures (section 2).

As concerns distance measures, the DTW-experiment showed that we cannot simply apply sequence/pattern matching to obtain good results in discovering and classification of these sub-sequences. This is a detriment to methods like Nearest Neighbour searches using typical distance (similarity) measures, but it also implies that an algorithm such as HTM struggles recognizing what should potentially be comparable sequences. We need to remind ourselves however, that after preprocessing into segments and feature extraction, KNN was not comparing sequences directly, but the feature vectors created from the underlying sequences. The feature extraction step has potentially improved the classifier's classification performance, as could be expected.

We have obtained mixed results from application of over-sampling and under-sampling methods to improve classification accuracy. Some interesting observations were made as to the various classifiers combined with the SMOTE oversampling method. A significant number of top scoring entries used SMOTE apart from combinations involving Extra-Trees classifier. Under-sampling on the other hand, has had less noticeable impact, which is somewhat surprising, given the extreme dominance of negative instances.

Due to time and resource constraints we could not possibly focus on finding the most optimal parameters for every step of the process which could be tuned. Filtering, oversampling, under-sampling are some examples where only one setting was used that could be further experimented with. The same can be said for many features produced with the TSfresh library – attempting to research and experiment with alternative parameter settings for each feature would be a major project in itself!

For HTM, we performed hundreds of tests using anomaly detection models with varying parameters to discover their sensitivity to changing activity patterns during migraine attacks. Models were allowed to build a "norm" for a certain number of timesteps and then set to report anomaly scores for the remaining data. The migraine attacks data when processed without applying timestamp information, purely as a stream of values, has given poor results with anomaly detection. This could be attributed to the fact that timestamp information adds an additional discriminative dimension to the data, by preserving the time-context of learned sequences. Such effect is quite intuitive and we would expect that it would be beneficial for anomaly detection in increasingly long monitoring periods. Longer periods, means that weekdays and weekends gain importance, something we could not reliably test with only 14 days' worth of data.

Anomaly detection has been used with the intention of discovering migraine attacks, but this unveiled an unsurprising fact in that an anomaly does not necessarily imply an attack. Any event that fails to reach some threshold of resemblance to the already known will be flagged. Furthermore, if the abnormal event repeats within a certain period, it will no longer be regarded as anomalous. This is due to the continuous learning capability of HTM, which can be tweaked. This brings us back to the problem of recognizing the event to ignore. Unlike the classification experiments, our HTM experiments were applied directly to the data sequence. More specifically, no window segments or features were extracted. HTM processes the data as temporal sequences by matching new patterns against those historically learned. In section 2 on similarity measures, we found that the migraine attack sequences respond poorly to distance and correlation based comparison. In this context, we find it plausible to assume that the reason that DTW sequence matching failed can be related to the poor performance of HTM. In summary, anomaly detection provides anonymous results, so its practicality for this task is questionable. Consequently, in context of RQ2, we conclude that anomaly detection with HTM is not recommended for this problem.

6.1 Problems

As we have pointed out in section 1.5, the bipolar long-term datasets did not become available in time to be used in this work. Consequently, the migraine-attacks dataset was used as: (a) it was collected using wearable actigraph, (b) measurements were continuously taken over a relatively long period, (c) the event of interest was labelled. The migraine data was assumed to be a fitting substitute for the task of working with detection of specific event over time, but cannot be considered for drawing any conclusions related to bipolar phase transitions prior to mood episodes.

Amount of data, specifically the absolute number of positive samples (attacks) turned out to be the biggest fundamental problem. The only solution here is to acquire more data as it becomes available. Additionally, we found reasons for scepticism about the accuracy of manual labelling, which we describe in more detail in the next section together with a suggestion for improvements.

The extreme class imbalance was a major challenge for the supervised classification part of the experiments, while it could be said that for anomaly detection it was the opposite. The problem encountered with our own implementation of over-sampling method can't be reliably judged before a much more significant number of positive samples can be examined. The underlying idea, certainly seems sound. The fact that differences between positive samples are apparently greater than between specially selected negative-positive pairs is something that should be validated with bigger datasets. In later stages, we found a flaw that in effect allowed accidental data leakage into the test set. Due to time constraints, we did not spend much time on correction and improvements of this method as we should and settled on using known methods.

Technical issues were encountered with both HTM and the TSFresh-package. In case of HTM, the discrepancy was between the so-called swarming model creation and usage of the model afterwards. Specifically, the "maxBoost" vs "Boosting" parameters were causing crashes. This and some failing unit tests, which could apparently "be ignored", according to information obtained from HTM forums. The TSFresh package has issues with parallelization on the AMD-based computer system that was used for running most of the classification experiments. We found that using version 0.5 rather than newer versions solved this issue.

6.2 Possible Improvements

Labelling

Something that has not been explicitly mentioned is attack reporting (labelling). The reliability of labelling attacks in the data should be questioned. Since the labels are based on patient-kept diaries the registered time of attack may be quite inaccurate, depending how the patients organize adding entries of the days' events. We have come upon this possibility after encountering an attack start that did not show any actual activity value (as recorded by the actigraph). The assumption is that some activity is registered in connection with the act of manually writing an entry in the diary to be registered by the actigraph at around the time of reported attack. Naturally, a patient could add the entry at a later time, but this is exactly why this entry might be inaccurate. This "later time" could be 10 minutes or 6 hours, in which case the accuracy of the patient's recollection could be questioned. Modern actigraphs come with a feature that allows inserting a timestamped label by the click of a button. We believe this could improve the reliability of labelling by simplifying this manual task for the patient.

Sampling Frequency

The effective sampling rate of the signal in the data is 1/60Hz (1 minute intervals). Internally, the signal is sampled at a constant rate of 0.32Hz by the actigraph, but then sum-aggregated into 1min intervals [70], which is the data we obtained. Naturally, this greatly reduces the information that we can possibly extract from the data. The reason can be shown using a simple example: given 2 sequences {1,1,1,1,21} & {5,5,5,5,5} with equal number of data points, we calculate their sums (25 & 25). If we assume that the elements represent samples before aggregation into a single value, as described above, we can see that in both cases our effective activity reading would be 25. The argument is that the pattern formed by these more frequent samples is lost, denying us the possibility to build a more accurate model. Higher frequency sampling is a requirement for accurate classification of activities, which when combined into sequences of activities should potentially produce much more useful information. Advantages of more finely grained data is increased information gain, more useful features, promotes recognition of individual activity patterns on the level of ADL's [80] and lessens 'blur' or noise which blurs the activity information. The main disadvantages are: memory requirement increase, processing power increase, increased battery power usage.

Deep Learning

Admittedly, there was some bias involved in the way hyper-parameters were pre-selected for the classification experiments. This was especially true for MLP (id.5, 8Appendix A) and

SVC (id.7, 8Appendix A). In both cases we intentionally limited the size of the hyper-parameter sets even though we would prefer to provide many more possible combinations. The reason stems from the fact that training these two algorithms is (time) expensive and our time resources were limited. In the early stages, deep learning frameworks (Keras & Tensorflow) were considered for inclusion in this work, as they are designed to handle ANN training very efficiently by utilizing GPUs (Graphical Processing Unit). This would eliminate the slow (Scikit Learn [20]) MLP-implementation but not the SVC, so we decided to defer Deep Learning for a later time. We know from unofficial and unpublished experiments that deep learning is an interesting direction to follow, even though our MLP results are not that good.

Sensor fusion.

A potential improvement on a higher level (i.e. the sensor hardware), a straightforward improvement might involve sensor fusion [68], in many ways similar to the sensor ensembles used in "Robust and Accurate Modelling Approaches for Migraine Per-Patient Prediction from Ambulatory Data" [74]. Fusion of sensor inputs intelligently combines data from several sensors thereby improving application or system performance. Combining input from multiple sensors corrects for the deficiencies of the individual sensors effectively improving accuracy and reliability. Every relevant input is a possible source of new features, correlations between series (or lack of such) may have a greatly positive influence on the effectiveness of classifiers. The disadvantages could be increased system cost, greater hardware requirements, higher memory usage and increased complexity.

7 Conclusion

We found specific combinations of preprocessing techniques, features and models that obtained promising results for potential implementation in a monitoring application. While the two best models use KNN and Voting ensemble, we found that Extra-Trees Forest classifier dominates the top results in general. We observed that the combination of extended feature set and feature selection consistently improves results, as does artefact removal and filtering to a smaller degree. SMOTE stands out positively from other class balancing methods, with the notable exception of combinations involving Extra-Trees. Additional fine-tuning of these combinations is possible and recommended as it may further improve classification performance.

We have shown that it is possible to detect actigraph patterns related to migraine attacks, with fair accuracy by using supervised learning. The methods used should be applicable in similar tasks. Assuming transitions prior to bipolar mood episodes can be proven to correlate with specific discernible patterns in the time/frequency domain, then these patterns can be detected, thereby effectively predicting the mood episodes.

Anomaly detection results were relatively poor, however more historic data should improve HTM's accuracy in detecting abnormal patterns. More importantly, HTM results suffer from ambiguity as to the context of the anomaly from noisy actigraph data. This seriously limits its utility for the task of detecting specific events like migraine attacks. For this reason, HTM anomaly detection is not recommended for this task.

The methods of labelling attacks during collection should be revised to lower the risk of inconsistencies. Finally, it is crucially important to note that the presented results are based on a very small positive sample size. In order to draw more reliable conclusions, much more data is recommended.

8 References

1. Patel, S., Park, H., Bonato, P., Chan, L., & Rodgers, M. (2012). A review of wearable sensors and systems with application in rehabilitation. *Journal of neuroengineering and rehabilitation*, *9*(1), 21. doi:10.1186/1743-0003-9-21

2. Pantelopoulos, A., & Bourbakis, N. G. (2010). A survey on wearable sensor-based systems for health monitoring and prognosis. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *40*(1), 1–12. doi:10.1109/TSMCC.2009.2032660

3. Chan, S. M. (2014). How innovation in mobile health is reshaping mental health. Retrieved May 26, 2017, from http://www.imedicalapps.com/2014/01/mobile-mental-health-behavioral-sensor-frameworks-schizophrenia/

4. Whelan, P., Machin, M., Lewis, S., Buchan, I., Sanders, C., Applegate, E., … Ainsworth, J. (2015). Mobile early detection and connected intervention to coproduce better care in severe mental illness. *Studies in Health Technology and Informatics*, *216*, 123–126. doi:10.3233/978-1-61499-564-7-123

5. Novák, D., Albert, F., & Španiel, F. (2014). Analysis of actigraph parameters for relapse prediction in bipolar disorder: a feasibility study. In *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference* (Vol. 2014, pp. 4972–4975). IEEE. doi:10.1109/EMBC.2014.6944740

6. Islam, S. K., Fathy, A., Wang, Y., Kuhn, M., & Mahfouz, M. (2014). Hassle-free vitals. *IEEE Microwave Magazine*, *15*(7), 525–533. doi:10.1109/MMM.2014.2356148

7. Deak, M. (2009). Use of Actigraphy in Neurological Patient Populations. *Neurol. Bull.*, *1*(1), 17–23. doi:10.7191/neurol

8. Morgenthaler, T., Alessi, C., Friedman, L., Owens, J., Kapur, V., Boehlecke, B., … Swick, T. J. (2007). Practice parameters for the use of actigraphy in the assessment of sleep and sleep disorders: an update for 2007. *Sleep*, *30*(4), 519–529. doi:10.1017/CBO9781107415324.004

9. Health, T. N. I. of M. (n.d.). NIMH » Bipolar Disorder. The National Institute of Mental Health. Retrieved from https://www.nimh.nih.gov/health/topics/bipolar-disorder/index.shtml

10. Janney, C. A., Fagiolini, A., Swartz, H. A., Jakicic, J. M., Holleman, R. G., & Richardson, C. R. (2014). Are adults with bipolar disorder active? Objectively measured physical activity and sedentary behavior using accelerometry. *Journal of Affective Disorders*, *152–154*(1), 498–504. doi:10.1016/j.jad.2013.09.009

11. Karam, Z. N., Provost, E. M., Singh, S., Montgomery, J., Archer, C., Harrington, G., & Mcinnis, M. G. (2014). Ecologically Valid Long-Term Mood Monitoring of Individuals With Bipolar Disorder Using Speech. *2014 Ieee International Conference on Acoustics, Speech and Signal Processing (Icassp)*, *2014*, 4858–4862. Retrieved from https://web.eecs.umich.edu/~baveja/Papers/Priori_ICASSP_final.pdf

12. Glenn, T., Whybrow, P. C., Rasgon, N., Grof, P., Alda, M., Baethge, C., & Bauer, M. (2006). Approximate entropy of self-reported mood prior to episodes in bipolar

disorder. *Bipolar disorders*, *8*(5 Pt 1), 424–9. doi:10.1111/j.1399-5618.2006.00373.x

13. About INTROMAT – Intromat. (n.d.). Retrieved May 10, 2017, from http://intromat.no/about/

14. Numenta.com • Leading the New Era of Machine Intelligence. (n.d.). Retrieved May 27, 2017, from https://numenta.com/

15. Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78. doi:10.1145/2347736.2347755

16. Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science (New York, N.Y.)*, *349*(6245), 255–60. doi:10.1126/science.aaa8415

17. What Is Big Data? - Blog. (n.d.). Retrieved May 31, 2017, from https://datascience.berkeley.edu/what-is-big-data/

18. Kuncheva, L. I. (2006). On the optimality of Naïve Bayes with dependent binary features. *Pattern Recognition Letters*, *27*(7), 830–837. doi:10.1016/j.patrec.2005.12.001

19. Brownlee, J. (n.d.). Parametric and Nonparametric Machine Learning Algorithms - Machine Learning Mastery. Retrieved May 20, 2017, from http://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/

20. scikit-learn: machine learning in Python — scikit-learn 0.18.1 documentation. (n.d.). Retrieved April 12, 2017, from http://scikit-learn.org/stable/

21. Bewick, V., Cheek, L., & Ball, J. (2005). Statistics review 14: Logistic regression. *Critical Care*, *9*(1), 112. doi:10.1186/cc3045

22. Auria, L., & Moro, R. A. (2008). Support Vector Machines (SVM) as a Technique for Solvency Analysis. Retrieved from www.diw.de

23. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. *Springer Texts in Statistics*. doi:10.1016/j.peva.2007.06.006

24. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. New York, NY: Springer New York. doi:10.1007/978-0-387-84858-7

25. Zughrat, A., Mahfouf, M., Yang, Y. Y., & Thornton, S. (2014). Support Vector Machines for Class Imbalance Rail Data Classification with Bootstrapping- based Over-Sampling and Under-Sampling. *IFAC Proceedings Volumes*, *47*(3), 8756–8761. doi:10.3182/20140824-6-ZA-1003.00794

26. Mitchell, T. M. (1997). *Machine Learning*. *Machine Learning* (Vol. 1). doi:10.1007/BF00116892

27. Dietterich, T. G. (2009). *Machine learning in ecosystem informatics and sustainability*. *IJCAI International Joint Conference on Artificial Intelligence* (Vol. 4). doi:10.1145/242224.242229

28. Bottou, L. (n.d.). Stochastic Gradient Descent Tricks. Retrieved from http://leon.bottou.org

29. Gradient Descent For Machine Learning - Machine Learning Mastery. (n.d.). Retrieved May 25, 2017, from http://machinelearningmastery.com/gradient-descent-for-machine-learning/

30. Dietterich, T. (2000). Multiple Classifier Systems. *Multiple Classifier Systems*, *1857*, 1–15. doi:10.1007/3-540-45014-9

31. Schapire, R. E. (2013). Explaining AdaBoost. In *Empirical Inference* (pp. 37–52). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-41136-6_5

32. Chen, C., Liaw, A., & Breiman, L. (2004). Using random forest to learn imbalanced data. *University of California, Berkeley*, *110*(1999), 1–12. doi:ley.edu/sites/default/files/tech-reports/666.pdf

33. Cutler, A., Cutler, D. R., & Stevens, J. R. (n.d.). Random Forests. doi:10.1007/978-1-4419-9326-7

34. Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42. doi:10.1007/s10994-006-6226-1

35. Cui, Y., Ahmad, S., & Hawkins, J. (2016). Continuous Online Sequence Learning with an Unsupervised Neural Network Model. *Neural Computation Massachusetts Institute of Technology*, *28*, 2474–2504. doi:10.1162/NECO_a_00893

36. Antic, S. D., Zhou, W. L., Moore, A. R., Short, S. M., & Ikonomu, K. D. (2010, November 1). The decade of the dendritic NMDA spike. *Journal of Neuroscience Research*. Wiley Subscription Services, Inc., A Wiley Company. doi:10.1002/jnr.22444

37. Hawkins, J., & Ahmad, S. (2016). Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. *Frontiers in Neural Circuits*, *10*(March), 23. doi:10.3389/fncir.2016.00023

38. Purdy, S. (2016). Encoding Data for HTM Systems. *arXiv*, 1602.05925. Retrieved from http://arxiv.org/abs/1602.05925

39. Ahmad, S., & Hawkins, J. (2015). Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory. Retrieved from http://arxiv.org/abs/1503.07469

40. George, D. (2006). Hierarchical temporal memory: theory and applications. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference*, *Suppl*, 6643. doi:10.1109/IEMBS.2006.260909

41. Wikipedia. (2016). Feature (machine learning) --- Wikipedia{,} The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Feature_(machine_learning)&oldid=702634191

42. Kotsiantis, S. B., Kanellopoulos, D., & Pintelas, P. E. (2006). Data preprocessing for supervised learning. *International Journal of Computer Science*, *1*(2), 111–117. doi:10.1080/02331931003692557

43. Khabou, M. A., & Parlato, M. V. (2013). Classification and feature analysis of actigraphy signals. *Conference Proceedings - IEEE SOUTHEASTCON*. doi:10.1109/SECON.2013.6567450

44. Banaee, H., Ahmed, M. U., & Loutfi, A. (2013). Data mining for wearable sensors in health monitoring systems: a review of recent trends and challenges. *Sensors (Basel, Switzerland)*, *13*(12), 17472–17500. doi:10.3390/s131217472

45. Géron, A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Retrieved from https://books.google.no/books?id=khpYDgAAQBAJ&pg=PA150&lpg=PA150&dq=polynomial+features+machine+learning&source=bl&ots=kKIxIRulq2&sig=JwfyhsHrrUY2eWcyFmxBoBR7FUk&hl=en&sa=X&ved=0ahUKEwj8ppXF4fvTAhVmJpoKHXa4BYc4ChDoAQhBMAU#v=onepage&q=polynomial features m

46. Heaton, J. (n.d.). An Empirical Analysis of Feature Engineering for Predictive Modeling. doi:10.1109/SECON.2016.7506650

47. Saeys, Y., Inza, I., & Larranaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, *23*(19), 2507–2517. doi:10.1093/bioinformatics/btm344

48. Hall, M. a., & Smith, L. a. (1998). Practical feature subset selection for machine learning. *Computer Science*, *98*, 181–191. Retrieved from http://researchcommons.waikato.ac.nz/handle/10289/1512

49. Fortmann-Roe, S. (2012). Accurately Measuring Model Prediction Error. Retrieved April 19, 2017, from http://scott.fortmann-roe.com/docs/MeasuringError.html

50. Brownlee, J. (n.d.). Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning - Machine Learning Mastery. Retrieved March 20, 2017, from http://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

51. Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, *30*(1), 27–38. doi:10.1016/j.patrec.2008.08.010

52. Akosa, J. S. (2017). Predictive Accuracy : A Misleading Performance Measure for Highly Imbalanced Data Classified negative, 1–12. Retrieved from http://support.sas.com/resources/papers/proceedings17/0942-2017.pdf

53. Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, *45*(4), 427–437. doi:10.1016/j.ipm.2009.03.002

54. Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. doi:10.1016/j.patrec.2005.10.010

55. Flight, L., & Julious, S. A. (n.d.). The Disagreeable Behaviour of the Kappa Statistic. Retrieved from https://www.sheffield.ac.uk/polopoly_fs/1.404095!/file/RSS_Poster_Laura_Flight_Final.pdf

56. McHugh, M. L. (2012). Interrater reliability: the kappa statistic. *Biochemia medica*, *22*(3), 276–82. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/23092060

57. Matthews Correlation Coefficient. (n.d.). Retrieved July 20, 2004, from https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Matthews_correlation_coefficient&id=757297687

58. Jurman, G., Riccadonna, S., & Furlanello, C. (2012). A comparison of MCC and CEN error measures in multi-class prediction. *PLoS ONE*, *7*(8), e41882. doi:10.1371/journal.pone.0041882

59. Ahmad, S., & Purdy, S. (n.d.). Real-Time Anomaly Detection for Streaming Analytics. Retrieved from https://arxiv.org/pdf/1607.02480.pdf

60. Lemaitre, G., Nogueira, F., & Aridas, C. K. (2016). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *CoRR*, *abs/1609.0*(17), 1–5. Retrieved from http://jmlr.org/papers/v18/16-365.html

61. Garc, V. (n.d.). Index of Balanced Accuracy : A Performance Measure for Skewed Class Distributions. Retrieved from http://repositori.uji.es/xmlui/bitstream/handle/10234/23961/33068.pdf;jsessionid=317C3F18725F892D93B54070F988A0D4?sequence=1

62. Longadge, R., Dongre, S. S., & Latesh, M. (2013). Class Imbalance Problem in Data Mining : Review. *International Journal of Computer Science and Network*, *2*(1).

63. Hoang, G., Bouzerdoum, A., & Lam, S. (2009). Learning Pattern Classification Tasks with Imbalanced Data Sets. In *Pattern Recognition* (pp. 193–208). InTech. doi:10.5772/7544

64. Liu, X., Wu, J., & Zhou, Z. (2009). Exploratory undersampling for class-imbalance learning. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, *39*(2), 539–50. doi:10.1109/TSMCB.2008.2007853

65. Wallace, B. C., Small, K., Brodley, C. E., & Trikalinos, T. A. (2011). Class imbalance, redux. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 754–763. doi:10.1109/ICDM.2011.33

66. Abd Elrahman, S. M., & Abraham, A. (2160). A Review of Class Imbalance Problem. *Journal of Network and Innovative Computing*, *1*, 332–340. Retrieved from http://www.softcomputing.net/jnic2.pdf

67. Martin, J. L., & Hakim, A. D. (2011, June). Wrist actigraphy. *Chest*. American College of Chest Physicians. doi:10.1378/chest.10-1872

68. Elmenreich, W. (2002). An introduction to sensor fusion. *Austria: Vienna University Of Technology*, (February), 1–28. Retrieved from http://www.vmars.tuwien.ac.at/documents/intern/805/elmenreich_sensorfusionintro.pdf

69. Krane-Gartiser, K., Henriksen, T. E. G., Morken, G., Vaaler, A., & Fasmer, O. B. (2014). Actigraphic assessment of motor activity in acutely admitted inpatients with bipolar disorder. *PLoS ONE*, *9*(2). doi:10.1371/journal.pone.0089574

70. Fasmer, O. B., Johnsen, E., Berle, J. Ø., Johnsen, E., Gjestad, R., Dilsaver, S., … Oedegaard, K. J. (n.d.). Objectively-measured motor activity patterns before, duringand after attacks of migraine: A pilot study, (1), 1–20.

71. Berle, J. O., Hauge, E. R., Oedegaard, K. J., Holsten, F., & Fasmer, O. B. (2010). Actigraphic registration of motor activity reveals a more structured behavioural pattern in schizophrenia than in major depression. *BMC research notes*, *3*, 149. doi:10.1186/1756-0500-3-149

72. Fasmer, O. B., Mjeldheim, K., Førland, W., Hansen, A. L., Dilsaver, S., Oedegaard, K. J., & Berle, J. Ø. (2015). Motor Activity in Adult Patients with Attention Deficit Hyperactivity Disorder, 474–482. doi:10.4306/pi.2015.12.4.474

73. Secil, Y., Unde, C., Beckmann, Y. Y., Bozkaya, Y. T., Ozerkan, F., & Basoglu, M. (2010). Blood pressure changes in migraine patients before, during and after migraine attacks. *Pain practice : the official journal of World Institute of Pain*, *10*(3), 222–227. doi:10.1111/j.1533-2500.2009.00349.x [doi]

74. Pagán, J., De Orbe, M. I., Gago, A., Sobrado, M., Risco-Martín, J. L., Vivancos Mora, J., … Ayala, J. L. (2015). Robust and Accurate Modeling Approaches for Migraine Per-Patient Prediction from Ambulatory Data. *Sensors*, *15*(7), 15419–15442. doi:10.3390/s150715419

75. Bruni, O., Russo, P. M., Violani, C., & Guidetti, V. (2004). Sleep and migraine: An actigraphic study. *Cephalalgia*, *24*(2), 134–139. doi:10.1111/j.1468-2982.2004.00657.x

76. Moore, P. J., Little, M. A., McSharry, P. E., Goodwin, G. M., & Geddes, J. R. (2014). Mood dynamics in bipolar disorder. *International Journal of Bipolar Disorders*, *2*(1), 11. doi:10.1186/s40345-014-0011-z

77. Ortiz, A., Bradler, K., Garnham, J., Slaney, C., & Alda, M. (2015). Nonlinear dynamics of mood regulation in bipolar disorder. *Bipolar Disorders*, *17*(2), 139–149. doi:10.1111/bdi.12246

78. Wang, J., Xian, H., Licis, A., Deych, E., Ding, J., McLeland, J., … Shannon, W. (2011). Measuring the impact of apnea and obesity on circadian activity patterns using functional linear modeling of actigraphy data. *Journal of Circadian Rhythms*, *9*(1), 11. doi:10.1186/1740-3391-9-11

79. Ellis, K., Godbole, S., Chen, J., Marshall, S., Lanckriet, G., & Kerr, J. (2013). Physical activity recognition in free-living from body-worn sensors. In *Proceedings of the 4th International SenseCam & Pervasive Imaging Conference on - SenseCam '13* (pp. 88–89). New York, NY, USA: ACM. doi:10.1145/2526667.2526685

80. Srinivasan, R., Chen, C., & Cook, D. J. (2010). Activity Recognition using Actigraph Sensor. *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data - SensorKDD '10*, 0–5.

81. Hemminki, S., Nurmi, P., & Tarkoma, S. (2013). Accelerometer-based transportation mode detection on smartphones. *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys '13*, 1–14. doi:10.1145/2517351.2517367

82. Cruz, R., Fernandes, K., Cardoso, J. S., & Costa, J. F. P. (2016). Tackling Class Imbalance with Ranking. In *International Joint Conference on Neural Networks (IJCNN)* (pp. 2182–2187). IEEE. doi:10.1109/IJCNN.2016.7727469

83. Wang, C., Viswanathan, K., Choudur, L., Talwar, V., Satterfield, W., & Schwan, K. (2011). Statistical techniques for online anomaly detection in data centers. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops* (pp. 385–392). doi:10.1109/INM.2011.5990537

84. Carús Candás, J. L., Peláez, V., López, G., Fernández, M. Á., Álvarez, E., & Díaz, G.

(2014). An automatic data mining method to detect abnormal human behaviour using physical activity measurements. *Pervasive and Mobile Computing*, *15*, 228–241. doi:10.1016/j.pmcj.2014.09.007

85. Kosorus, H., Honigl, J., & Kung, J. (2011). Using R, WEKA and RapidMiner in Time Series Analysis of Sensor Data for Structural Health Monitoring. *2011 22nd International Workshop on Database and Expert Systems Applications*, 306–310. doi:10.1109/DEXA.2011.88

86. Nanopoulos, A., Alcock, R., & Manolopoulos, Y. (n.d.). Feature-based Classiication of Time-series Data. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.9555&rep=rep1&type=pdf

87. Fasmer, O. B., Johnsen, E., Berle, J., Øystein, Johnsen, E., Gjestad, R., … Oedegaard, K. J. (n.d.). Objectively-measured motor activity patterns before, during and after attacks of migraine: A pilot study, (1), 1–20.

88. Stojanova, D. (n.d.). CONSIDERING AUTOCORRELATION IN PREDICTIVE MODELS. Retrieved from http://kt.ijs.si/theses/phd_daniela_stojanova.pdf

89. Autocorrelation and Partial Autocorrelation - MATLAB & Simulink - MathWorks Nordic. (n.d.). Retrieved May 25, 2017, from https://se.mathworks.com/help/econ/autocorrelation-and-partial-autocorrelation.html

90. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., … Keogh, E. (2013). Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. *Transactions on Knowledge Discovery from Data (TKDD*, *7*(3), 3047–3051. doi:10.1145/2500489

91. Cassisi, C., Montalto, P., Aliotta, M., Cannata, A., & Pulvirenti, A. (2012). Similarity Measures and Dimensionality Reduction Techniques for Time Series Data Mining. *Advances in Data Mining Knowledge Discovery and Applications*, (May 2017). doi:10.5772/49941

92. Salvador, S., & Chan, P. (2007). FastDTW : Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, *11*(5), 561–580. Retrieved from https://pdfs.semanticscholar.org/05a2/0cde15e172fc82f32774dd0cf4fe5827cad2.pdf

93. Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, *97*(1–2), 245–271. doi:10.1016/S0004-3702(97)00063-5

94. Maximilian Christ, Nils Braun, J. N. (2017). TSFresh. Retrieved from https://media.readthedocs.org/pdf/tsfresh/latest/tsfresh.pdf

95. Tukey (tapered cosine) window - MATLAB tukeywin - MathWorks Nordic. (n.d.). *MathWorks*. Retrieved May 9, 2017, from https://se.mathworks.com/help/signal/ref/tukeywin.html#f8-511559

96. Hodrick-Prescott filter for trend and cyclical components - MATLAB hpfilter - MathWorks Nordic. (n.d.). *Mathworks*. Retrieved May 9, 2017, from https://se.mathworks.com/help/econ/hpfilter.html

Appendix A Classifier hyper-parameter sets

Note: The VotingClassifer is an ensemble of LogisticRegression (lr), RandomForest (rf) and KNN (knn), all with individual hyper-parameters (prefixed).

ID	Name	Parameter: [value set]
0	Gaussian NB	{}
1	Logistic Regression	'C': [0.001, 0.01, 0.1, 1, 5, 10], 'class_weight': [{1:3}, {1:15}, {1:50}, {1:500}, 'balanced'], 'intercept_scaling': [0.1, 0.01, 1, 5, 10], 'solver': ['lbfgs', 'liblinear', 'sag']}
2	SGD Classifier	'alpha': [0.0001, 0.001, 0.1, 0.5, 1], 'class_weight': [{1:3}, {1:15}, {1:50}, {1:500}, 'balanced'], 'eta0': [0.01, 0.001, 0.5], 'learning_rate': ['constant', 'optimal', 'invscaling'], 'loss': ['modified_huber', 'squared_hinge', 'perceptron'], 'penalty': ['none', 'l1', 'l2', 'elasticnet']},
3	Extra Trees	'n_estimators': [15, 30, 50], 'criterion': ['gini', 'entropy'], 'max_depth': [8, 16, 32, None], 'class_weight': [{1:3}, {1:15}, {1:50}, {1:500}, 'balanced'], 'max_features': ['auto', 0.75, 0.33]},
4	Random Forest	'n_estimators': [15, 30, 50], 'criterion': ['gini', 'entropy'], 'max_depth': [8, 16, 32, None], 'class_weight': [{1:3}, {1:15}, {1:50}, {1:500}, 'balanced'], 'max_features': ['auto', 0.75, 0.33]},
5	MLP	'alpha': [0.001, 0.01, 0.1, 0.5], 'solver': ['lbfgs'], # 'sgd', 'adam'], 'activation': ['tanh', 'relu'], 'learning_rate': ['adaptive', 'invscaling'], 'hidden_layer_sizes': [(240,40,), (60,20,10,), (500,)]},
6	AdaBoost	'n_estimators': [15, 50, 80], 'base_estimator__criterion': ['gini', 'entropy'], 'base_estimator__splitter': ['best', 'random'], 'base_estimator__max_depth': [4, 8, 16, 32], 'base_estimator__class_weight': [{1:5}, {1:50}, {1:500}, 'balanced'], 'learning_rate': [0.5, 1.0, 2.0]},
7	SVC	'decision_function_shape': ['ovr'], 'C': [0.01, 0.1, 0.5, 1, 10], 'gamma': [0.01, 0.1, 1, 10], 'class_weight': [{1:3}, {1:50}, {1:500}, 'balanced'], 'kernel': ['linear', 'rbf']},

8	Voting Classifier	'voting': ['soft'], #, 'hard'], 'lr__C': [0.1, 1.0, 10.0], 'lr__class_weight': [{1:15}, {1:500}, 'balanced'], 'rf__n_estimators': [10, 30], 'rf__class_weight': [{1:15}, {1:500}, 'balanced'], 'knn__weights': ['distance', 'uniform'], 'knn__n_neighbors': [2, 4]},
9	KNN	'weights': ['distance', 'uniform'], 'p': [1, 2, 3, 4], 'metric': ['minkowski'], 'leaf_size': [30, 5], 'n_neighbors': [1, 2, 3, 4, 5, 6]},

Appendix B HTM base model settings

```
MODEL_PARAMS={
            'aggregationInfo': {
        'days': 0,
        'fields': [],
        'hours': 0,
        'microseconds': 0,
        'milliseconds': 0,
        'minutes': 0,
        'months': 0,
        'seconds': 0,
        'weeks': 0,
        'years': 0
},
'model': 'CLA',
'modelParams': {
        'anomalyParams': {
        u'anomalyCacheRecords': None,
        u'autoDetectThreshold': None,
        u'autoDetectWaitRecords': 5030
},
'clParams': {
        'alpha': 0.0381314,
        'verbosity': 0,
        'regionName': 'SDRClassifierRegion',
        'steps': '1'
},
'inferenceType': 'TemporalAnomaly',
'sensorParams': {
        'encoders': {
        '_classifierInput': {
        'classifierOnly': True,
        'clipInput': True,
        'fieldname': 'value',
        'maxval': 5000,
        'minval': 0,
        'n': 400,
        'name': '_classifierInput',
        'type': 'ScalarEncoder',
        'w': 21
},
u'value': {
        'clipInput': True,
        'fieldname': 'value',
        'maxval': 4000,
        'minval': 0,
        'n': 400,
        'name': 'value',
        'type': 'ScalarEncoder',
        'w': 21
},
u'timestamp_dayOfWeek': None,
u'timestamp_timeOfDay': {
        'fieldname': 'timestamp',
        'name': 'timestamp',
        'timeOfDay': (21, 2.0),
        'type': 'DateEncoder'
},


 u'timestamp_weekend': None
},
'sensorAutoReset': None,
'verbosity': 0
},
'spEnable': True,
'spParams': {
        'columnCount': 2048,
        'globalInhibition': 1,
        'inputWidth': 0,
        'numActiveColumnsPerInhArea': 40,
```

```
        'potentialPct': 0.8,
        'seed': 1956,
        'spVerbosity': 0,
        'spatialImp': 'cpp',
        'synPermConnected': 0.1,
        'synPermActiveInc': 0.0001,
        'synPermInactiveDec': 0.0005,
        'maxBoost': 0.0,

    },
    'tpEnable': True,
    'tpParams': {
        'activationThreshold': 12,
        'cellsPerColumn': 32,
        'columnCount': 2048,
        'globalDecay': 0.0,
        'initialPerm': 0.21,
        'inputWidth': 2048,
        'maxAge': 0,
        'maxSegmentsPerCell': 128,
        'maxSynapsesPerSegment': 32,
        'minThreshold': 10,
        'newSynapseCount': 20,
        'outputType': 'normal',
        'pamLength': 1,
        'permanenceDec': 0.1,
        'permanenceInc': 0.1,
        'seed': 1960,
        'temporalImp': 'cpp',
        'verbosity': 0
    },
    'trainSPNetOnlyIfRequested': False
    },
    'predictAheadTime': None,
    'version': 1
    }
```

Fields highlighted in **RED** were manipulated during the experiment parameter tuning phase. These values could differ from values presented in this snapshot. Remaining base models are attached as files.

Appendix C Classification result table codes

Code	***Description***
Id	Excel table entry id
test id	Sequentially numbered experiment batch (per summary file)
alg id	Algorithm identification number (see 8Appendix A)
acc	Classification accuracy score
mcc	Matthews Correlation Coefficient
kap	Cohens Kappa statistic
auc	Area under the ROC curve
geo	Geometric mean
iba	Index of balance accuracy
TN	True Negatives
FP	False Positives
FN	False Negatives
TP	True Positives
Pr	Precision
re	Recall
f1	F1-measure
sp	Support (number of instances in evaluation set)
pca	PCA setting (0 for None)
poly	Polynomial feature order value (0 for None)
ksel	K-select setting (0 for None)
exs	Total examples created
feats	Total features created
tvars	Total target class labels
imb	Class balancing method applied (or None)
time	Execution time for the batch (i.e. test id)
filter	Filter flag
set	Extended feature set flag

Appendix D Extended feature set

Overview of features calculated for the extended feature set. Several of these results in more than one feature columns, depending on the parameters passed. This overview is a modified transcript of the more detailed descriptions found in the “TSFresh” manual [94].

Feature name	Short description
abs_energy	Returns the absolute energy of the time series which is the sum over the squared values
absolute_sum_of_changes	Returns the sum over the absolute value of consecutive changes in the series x
approximate_entropy(x, m, r)	Implements a vectorised Approximate entropy algorithm.
ar_coefficient	This feature calculator fits the unconditional maximum likelihood of an autoregressive AR(k) process. The k parameter is the maximum lag of the process
augmented_dickey_fuller	The Augmented Dickey-Fuller test is a hypothesis test which checks whether a unit root is present in a time series sample. This feature calculator returns the value of the respective test statistic.
autocorrelation	Calculates the lag autocorrelation of a lag value of lag.
binned_entropy	First bins the values of x into max-bins equidistant bins. Then calculates the value of where p_k is the percentage of samples in bin k.
count_above_mean	Returns the number of values in x that are higher than the mean of x
count_below_mean	Returns the number of values in x that are lower than the mean of x
cwt_coefficients	Calculates a Continuous wavelet transform for the Ricker wavelet, also known as the “Mexican hat wavelet”. This feature calculator takes three different parameters: widths, coeff and w. The feature calculator takes all the different widths arrays and then calculates the cwt one time for each different width array. Then the values for the different coefficient for coeff and width w are returned.
fft_coefficient	Calculates the Fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast Fourier transformation algorithm
first_location_of_maximum	Returns the first location of the maximum value of x. The position is calculated relatively to the length of x.
first_location_of_minimum	Returns the first location of the minimal value of x. The position is calculated relatively to the length of x.
friedrich_coefficients(x, c, param)	Coefficients of polynomial, which has been fitted to the deterministic dynamics of Langevin model
has_duplicate	Checks if any value in x occurs more than once
has_duplicate_max	Checks if the maximum value of x is observed more than once
has_duplicate_min	Checks if the minimal value of x is observed more than once
index_mass_quantile	Those apply features calculate the relative index i where q% of the mass of the time series x lie left of i. For example, for q = 50% this feature calculator will return the mass centre of the time series
Kurtosis	Returns the kurtosis of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).
large_number_of_peaks	Checks if the number of peaks is higher than n.

large_standard_deviation	Boolean variable denoting if the standard dev of x is higher than 'r' times the range = difference between max and min of x.
last_location_of_maximum	Returns the relative last location of the maximum value of x. The position is calculated relatively to the length of x.
last_location_of_minimum	Returns the last location of the minimal value of x. The position is calculated relatively to the length of x.
length(x)	Returns the length of x
longest_strike_above_mean	Returns the length of the longest consecutive subsequence in x that is bigger than the mean of x
longest_strike_below_mean	Returns the length of the longest consecutive subsequence in x that is smaller than the mean of x
max_langevin_fixed_point(x, r, m)	Largest fixed point of dynamics estimated from polynomial h(x), which has been fitted to the deterministic dynamics of Langevin model
maximum(x)	Calculates the highest value of the time series x
mean(x)	Returns the mean of x
mean_abs_change	Returns the mean over the absolute differences between subsequent time series values which is
mean_abs_change_quantiles	First fixes a corridor given by the quantiles ql and qh of the distribution of x. Then calculates the average absolute value of consecutive changes of the series x inside this corridor. Think about selecting a corridor on the Y-Axis and only calculating the mean of the absolute change of the time series inside this corridor.
mean_autocorrelation	Calculates the average autocorrelation taken over different all possible lags (1 to length of x)
mean_change	Returns the mean over the absolute differences between subsequent time series values which is
mean_second_derivate_central	Returns the mean value of a central approximation of the second derivative
median(x)	Returns the median of x
minimum(x)	Calculates the lowest value of the time series x.
number_cwt_peaks	This feature calculator searches for different peaks in x. To do so, x is smoothed by a Ricker-wavelet and for widths ranging from 1 to n. This feature calculator returns the number of peaks that occur at enough width scales and with sufficiently high Signal-to-Noise-Ratio (SNR)
number_peaks	Calculates the number of peaks of at least support n in the time series x. A peak of support n is defined as a subsequence of x where a value occurs, which is bigger than its n neighbours to the left and to the right.
percentage_of_reoccurring_datapoints_to_all_datapoints	Returns the percentage of unique values, that are present in the time series more than once.
percentage_of_reoccurring_values_to_all_values	Returns the ratio of unique values, that are present in the time series more than once.
Quantile	Calculates the q quantile of x. This is the value of x greater than q% of the ordered values from x.
range_count(x, min, max)	Count observed values within the interval [min, max).
ratio_value_number_to_time_series_length	Returns a factor which is 1 if all values in the time series occur only once, and below one if this is not the case.
sample_entropy(x)	Calculate and return sample entropy of x.

Skewness	Returns the sample skewness of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).
spkt_welch_density	This feature calculator estimates the cross power spectral density of the time series x at different frequencies. To do so, the time series is first shifted from the time domain to the frequency domain. The feature calculator returns the power spectrum of the different frequencies.
standard_deviation	Returns the standard deviation of x
sum_of_reoccurring_data_points	Returns the sum of all data points, that are present in the time series more than once.
sum_of_reoccurring_values	Returns the sum of all values, that are present in the time series more than once.
sum_values(x)	Calculates the sum over the time series values
symmetry_looking	Boolean variable denoting if the distribution of x *looks symmetric*.
time_reversal_asymmetry_statistic	This function calculates the value of $\mathbb{E}[L^2(X)^2 \cdot L(X) - L(X) \cdot X^2]$ where E is the mean and L is the lag operator.
value_count(x, value)	Count occurrences of *value* in time series x.
Variance	Returns the variance of x
variance_larger_than_standard_deviation	Boolean variable denoting if the variance of x is greater than its standard deviation. Is equal to variance of x being larger than 1

Appendix E Top scorers: raw output

Winner by AUC score: (source file: Results_200x25_basic_filter.txt)

```
pre/post: 25/0  win/stride: 200/25  label:start
filename: stm_sb_modified.csv
datetime: 2017-05-10 19:14:23.374000
pca_target: 0       poly degree: 0      kselect: 20
Imbalance: SMOTE(k=None, k_neighbors=5, kind='regular', m=None, m_neighbors=10,
n_jobs=1,
   out_step=0.5, random_state=None, ratio='auto', svm_estimator=None)
Extended target (count):  25141 325
('Total : Processed (count): ', (998L, 22L), 13)
Final feature (count):  (998L, 22L)

...

Running GridSearchCV for KNeighborsClassifier.
Fitting 5 folds for each of 96 candidates, totalling 480 fits
BEST: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=4, n_neighbors=2, p=2,
           weights='uniform')
        ACC    MCC    KAP    AUC    GEOM   IBA
       0.940  0.413  0.292  0.970  0.969  0.945
[[278  18]
 [  0   4]]
              precision    recall  f1-score   support

          0       1.00      0.94      0.97       296
          1       0.18      1.00      0.31         4

avg / total       0.99      0.94      0.96       300

...
```

Filter	Yes
Window size	200
Stride	25
Feature set	basic
PCA	0
polynomial	0
k-select	20
imbalance	SMOTE(k_neighbors=5, m_neighbors=10, out_step=0.5)
Classifier	KNeighborsClassifier
Leaf_size	30
Metric	Minkowski
N_neighbors	2
P	2
weights	Uniform

Winner by MCC score: (source file: Results_200x25_tsfresh_filter.txt)

```
pre/post: 25/0  win/stride: 200/25  label:start
filename: stm_sb_modified.csv
datetime: 2017-05-09 14:09:56.672000
pca_target: 0        poly degree: 0      kselect: 80
Imbalance: SMOTE(k=None, k_neighbors=5, kind='regular', m=None, m_neighbors=10,
n_jobs=1,
   out_step=0.5, random_state=None, ratio='auto', svm_estimator=None)
Extended target (count):  25141 325
('Total : Processed (count): ', (998L, 220L), 13)
Final feature (count):  (998L, 220L)

...
Running GridSearchCV for VotingClassifier.
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
BEST: VotingClassifier(estimators=[('lr', LogisticRegression(C=10.0,
class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=4, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)), ('rf',
...wski',
           metric_params=None, n_jobs=4, n_neighbors=2, p=2,
           weights='distance'))],
         n_jobs=4, voting='soft', weights=[1, 1, 1])
        ACC    MCC    KAP    AUC    GEOM   IBA
       0.993  0.705  0.664  0.750  0.707  0.475
[[296   0]
 [  2   2]]
             precision    recall  f1-score   support

        0.0       0.99      1.00      1.00       296
        1.0       1.00      0.50      0.67         4

avg / total       0.99      0.99      0.99       300
```

Filter	Yes
Window size	200
Stride	25
Feature set	extended
PCA	0
polynomial	0
k-select	80
imbalance	SMOTE(k_neighbors=5, m_neighbors=10, out_step=0.5)
Classifier	VotingClassifier
Ensemble	LogisticRegression, RandomForest, KNeighborsClassifier

Appendix F Libraries and tools

Python 2.7 was used for the majority of the code. Additionally, some early preprocessing and exploratory analysis was done using the R-language with R-Studio. All our code and result files are available at https://github.com/tomhvl/migraine-attacks/

The following external Python libraries were used in this project:

Statsmodels	Classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. http://www.statsmodels.org/stable/install.html
Pandas	BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools. http://pandas.pydata.org/pandas-docs/stable/install.html#
Numpy	The fundamental package for scientific computing with Python. Installs with Scikit-learn or from: https://www.scipy.org/scipylib/download.html
Imblearn	Package offering a number of re-sampling techniques commonly used in datasets showing strong between-class imbalance. https://github.com/scikit-learn-contrib/imbalanced-learn
Scikit-Learn	Python module for machine learning built on top of SciPy and distributed under the 3-Clause BSD license. https://github.com/scikit-learn/scikit-learn
FastDTW	Fast, linear time approximation algorithm for Dynamic Time Warping. https://github.com/rmaestre/FastDTW
MatplotLib	Python 2D plotting library. https://matplotlib.org/users/installing.html
NuPIC	NuPIC is a machine intelligence platform that implements the HTM learning algorithms. HTM is a detailed computational theory of the neocortex. distribution found at: https://github.com/numenta/nupic
tsfresh	Contains many feature extraction methods. https://github.com/blue-yonder/tsfresh