Methods and tools for mining multivariate time series

De Gouveia da Costa Cachucho, R.E.

Chapter 4

Mining Multivariate Time Series with Mixed Sampling Rates

Ricardo Cachucho, Marvin Meeng, Ugo Vespier, Siegfried Nijssen, Arno Knobbe

in Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp), 2014

Abstract

Fitting sensors to humans and physical structures is becoming more and more common. These developments provide many opportunities for ubiquitous computing, as well as challenges for analyzing the resulting sensor data. From these challenges, an underappreciated problem arises: modeling multivariate time series with mixed sampling rates. *Although mentioned in several application papers using sensor systems, this problem has been left almost unexplored, often hidden in a pre-processing step or solved manually as a one-pass procedure (feature extraction/construction). This leaves an opportunity to formalize and develop methods that address mixed sampling rates in an automatic fashion.*

We approach the problem of dealing with multiple sampling rates from an aggregation perspective. We propose Accordion, a new embedded method that constructs and selects aggregate features iteratively, in a memory-conscious fashion. Our algorithms works on both classification and regression problems. We describe three experiments on real-world time series datasets.

4.1 Introduction

This paper presents a practical modeling task in the field of multivariate time series analysis, and algorithms to solve this task. In real-world applications involving time series, specifically those produced by multiple sensors, one is often confronted with the challenge of analyzing data captured at various sampling rates. This might occur when one wants to include sensors that measure processes at various rates, for example the vibration (high sampling rate) and temperature (low rate) of a large windmill. In this paper, we analyze a specific instantiation of such a problem, where the aim is to model a target time series, that is captured at (much) lower rates than the remaining series. To model the target series in terms of the remaining ones, we will somehow have to 'slow down' the high-frequency measurements in order to match the target.

As a motivating example, consider the problem of *activity recognition*. In this problem, the task is to classify a person's activities into a finite set of classes, typically at a fairly slow rate. The activity will be predicted using body-worn or environmental sensors, for example measuring physiological parameters (e.g. heart rate), acceleration or position in space. The practical problem here is that modern sensors tend to measure at high sampling rates (typically 1 Hz or higher), whereas activity is registered at much lower rates (e.g. once every 60 seconds). Therefore, each period to be classified is described by many measurements (per sensor). The most obvious solution is to combine multiple measurements into a single value characterising the period, for example, the average heart rate or the highest acceleration experienced over this period.

In this paper, we approach this challenge from an aggregation perspective: for a given sensor and a given time interval (a window of data), an aggregate function will summarize a sequence of sensor readings into a single numeric value. An aggregate feature is composed of three main components: a sensor measuring at high rates (the *predictor*), a window over which values of the sensor are aggregated, and finally an *aggregate function* that combines these values into a single outcome. Assuming a target series sampled at low frequency $f_{\mathbf{r}}$ and the remaining time series at higher frequency $f_{\mathbf{S}}$, the proportion $f_{\mathbf{S}}/f_{\mathbf{r}}$ denotes the number of measurements per sensor that correspond to a single target value. To allow for phenomena that involve unknown degrees of integration over time, our algorithm will be allowed to consider windows both longer and shorter than $f_{\mathbf{S}}/f_{\mathbf{r}}$.

To illustrate how the optimal window size may be somewhat larger or smaller than the proportion $f_{\mathbf{S}}/f_{\mathbf{r}}$, consider the challenge of modeling a person's sleep quality (one of our applications mentioned in the experimental section) from various sensors, both body-worn and placed in the environment. In this case, 24 hours of sensor data naturally correspond to a single target value describing the sleep quality of one night. However, one can imagine that a feature capturing the amount of strenuous activity during the four hours prior to sleep might play an important role, which corresponds to a window size of only 1/6th of the 'natural' window. Similarly, windows covering more than 24 hours are imaginable, such as those related to the nutrition over the last 48 hours. Clearly, a fixed window size based on the mentioned proportion will not guarantee optimal results, and we will have to include and optimize the size of the window as a parameter in the definition of features.

Although a feature construction step (even including aggregation) is the backbone of many activity recognition projects [25], all too often this step is presented as a one-pass process [6, 69, 58, 75], such that only a fixed set of features becomes available for the actual modeling step. The resulting features are static and constructed manually, either based on some domain knowledge about the physics involved, or by making default choices. It is not hard to imagine that this step is, in fact, the result of several iterations of trial and error. Moreover, this fixed set is required to be relatively small, for reasons of memory or storage. The iterative method we propose, Accordion, is an embedded approach that does both feature construction (building candidate aggregate features), and feature selection automatically, allowing for features to be created dynamically during the search process.

At each iteration in the search process, Accordion transforms high frequency predictors into a set of candidate aggregate features at the lower frequency of the target, searching for the best combination of the components that compose an aggregate feature. From this set of aggregate features, only the most promising candidate feature is selected and materialized, in a greedy fashion. Therefore we categorize our algorithms as *memory-conscious*. With the dynamic construction of features proposed here, we aim to solve both the issue of choosing the right features and estimating their parameters, as well as the varying requirements for the informative features that occur while modeling the data (for example, further down in a decision tree). In order to do so, the feature construction and selection steps are closely tied with the final modeling process, in both the regression and classification setting. Inspired by Brush's challenge of enhancing reproducibility and clarity [12],

our algorithms and activity recognition datasets are made available[1].

When thinking of the aggregate feature construction possibilities, it is good to note that the search space is potentially very large, due to the choices of sensor, aggregate function and window size (which may vary substantially, as noted). Therefore, we search for (candidate) aggregate features heuristically. We propose algorithms that consider a feasible set of candidate features by a) limiting the actual choice of aggregate functions to a small set (*min*, *max*, *avg*, ...), b) performing a moderate search over the possible window sizes, and c) selecting the final aggregate features at different degrees of greediness. On top of these choices, we tackle the potentially large size of the final dataset by materializing only the selected features.

In general, we distinguish between two types of applications, one where the slow target series is numeric, and we are effectively dealing with a *regression* model, and one where the target is nominal and we need a *classification* model. The feature selection algorithm works differently for either setting, but the essence of constructing sets of candidate features using aggregation is identical. The main difference between the two versions is the kind of modeling they mimic: in the regression case, the feature construction algorithm effectively builds a linear model in a greedy fashion. In the case of classification, we construct a decision tree of aggregate features along the lines of C4.5 [80].

The main contributions of this paper are as follows:

- Present and formalize a common task in the modeling of multivariate time series, related to the target being measured at a lower rate than the remaining series.
- Propose an embedded algorithm for the proposed task, that dynamically constructs, selects and models (all in one solution), using a manageable set of aggregate features in the contexts of both classification and regression.
- Describe how both algorithms are memory-conscious, materializing only a limited set of aggregate features, and potentially increasing the possible search space for good features.
- Algorithm implementation and datasets are made available to the research community.

[1] `http://www.liacs.nl/~cachucho/publications.html`

4.2 Preliminaries

4.2.1 Multivariate Time Series with Mixed Sampling Rates

The data we consider is assumed to come from *sensor systems*. We assume that our sensor system $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_p, \mathbf{r}\}$ consists of $p+1$ sensors. The first p sensors will act as *predictors*, while the last sensor $\mathbf{r}$, the *response*, will be treated as the target sensor that we wish to predict or explain. $|\mathbf{s}|$ and $|\mathbf{r}|$ indicate the length (number of data points) of $\mathbf{s}$ and $\mathbf{r}$, respectively. While the domain for the predictors is always the set of real numbers $\mathbb{R}$, the domain of $\mathbf{r}$ is either $\mathbb{R}$ (regression setting), or a finite set of classes (classification setting).

We assume that all sensors register measurements synchronously and at the same fixed sampling rate, *except* for the response, which is registering at a lower sampling rate. We also assume that the predictor sampling rate is an integer $q > 1$ multiple of the sampling rate of the response: $f_{\mathbf{S}} = q \cdot f_{\mathbf{r}}$. This leads to the following definition.

Definition 6. *A time series dataset with mixed sampling rates is assumed to consist of:*

- *A set of time series* $\mathbf{S}$*, representing the predictor variables, where* $\mathbf{S}$ *is materialized as a matrix of size* $|\mathbf{s}| \times p$*. Each time series* $\mathbf{s}$ *in* $\mathbf{S}$ *is a vector of real numbers, where* $\mathbf{s}_i, i = 1, \ldots, |\mathbf{s}|$ *is the* i*th element of* $\mathbf{s}$*.*
- *A time series* $\mathbf{r}$*, representing the response variable. This time series has a length of* $|\mathbf{r}| = |\mathbf{s}|/q$*, where* $q \in \mathbb{N}^+ \wedge q > 1$*; the* i*th element in* $\mathbf{r}$ *is assumed to have been measured at the same time as the* $i \cdot q$*th element in* $\mathbf{s}$*.*

Note that this implies that the measurements of $\mathbf{S}$ and $\mathbf{r}$ do not start at the same time (see Figure 4.1).

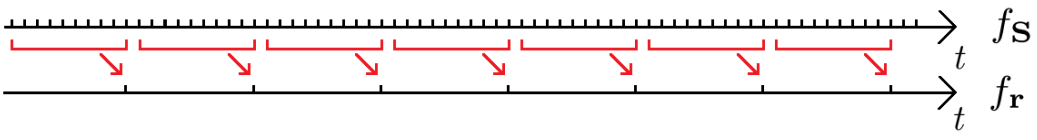

Figure 4.1: Relation between high ($f_{\mathbf{S}}$) and low ($f_{\mathbf{r}}$) sampling rates.

4.2.2 Feature Construction with Mixed Sampling Rates

As discussed, our aim is to model the *response variable* in terms of the *predictors*, over time. As the two are sampled at different rates, we will 'slow down' the high frequency measurements of $\mathbf{S}$ using aggregate functions, and turn them into features that are available at the sampling rate of $\mathbf{r}$. In other words, we will be taking a *feature construction* approach. In order to transform the high frequency measurements into lower frequency ones, we employ the notion of a *window*:

Definition 7 (Window)**.** *Given a window length w and an index $1 \leq w \leq i \leq |\mathbf{s}|$ in a predictor time series $\mathbf{s}$, a window of length w at index i consists of the time series of measurements $\mathbf{s}[i-w+1,\ldots,i] = \mathbf{s}_{i-w+1}, \mathbf{s}_{i-w+2}, \ldots, \mathbf{s}_i$.*

For a response series measured q times as slow as the predictors, it could make sense to choose w such that $w = q$. Figure 4.1 depicts such a situation. However, experimental evaluation reveiled that this choice may not always be optimal. Both window lengths $w > q$ and $w < q$ could also be argued for. Therefore, we simply assume that $w \in \mathbb{N}^+$. Note that when $w > q$, each consecutive window will have the following fraction of overlap: $1 - q/w$.

We will employ *aggregate functions* to summarize the measurements in a window into a single value. An aggregate function $a \in A$ takes as input a time series of measurements $\mathbf{m}$, and produces a single numeric value $a(\mathbf{m}) \in \mathbb{R}$. The fixed set of aggregate functions $A = \{min, max, avg, \ldots\}$ we use will be described in more detail in the next section.

We can now define aggregate features as follows.

Definition 8 (Aggregate Feature)**.** *Given a choice of window size w, an aggregate function a and a predictor time series $\mathbf{s} \in \mathbf{S}$, the aggregate feature $\boldsymbol{af}_{\mathbf{s},a,w}$ is a vector of length $|\mathbf{s}|/q$, defined as follows:*

$$\boldsymbol{af}_{\mathbf{s},a,w}[i'] = a(\mathbf{s}[\max(1, i' \cdot q - w + 1), \ldots,' \cdot q])$$

where $1 \leq i' \leq |\mathbf{s}|/q$.

An aggregate feature for a given dataset can hence be specified by a tuple of parameters $(\mathbf{s}, a, w)$. Sometimes we will refer to these features without reference to their parameters, just as a generic aggregate feature $\mathbf{f}$.

A set of aggregate features $\mathcal{F}$ together with the vector of response values $\mathbf{r}$ can be used to create a new data matrix. Each aggregate feature corresponds to a column of this matrix. The number of rows in this matrix corresponds exactly to the length of $\mathbf{r}$, $|\mathbf{r}| = |\mathbf{s}|/q$.

More formally, a data matrix $\mathbf{S}'$ of dimension $|\mathbf{r}| \times (|\mathcal{F}| + \mathbf{1})$ is obtained, where $\mathbf{S}'_{t,f}$ is the feature calculated for the tth target instance using the fth feature descriptor in $\mathcal{F}$.

4.2.3 Problem Statement

Our main task is to find good aggregate features for time series datasets with mixed sampling rates. More formally, we assume we are given a time series dataset as introduced in Definition 6, as well as a function $score(\mathcal{F}, \mathbf{r})$ that can evaluate the quality of a set of features with respect to response variable $\mathbf{r}$. The task is to find a set of aggregate features $\mathcal{F}$, such that each feature is described by:

- A predictor time series $\mathbf{s} \in \mathbf{S}$.
- An aggregate function $a \in A$.
- A window size w.

Furthermore, the feature set $\mathcal{F}$ should optimize the scoring function $score(\mathcal{F}, \mathbf{r})$. Scoring functions in this paper can be based on regression or classification models. The details of this will be discussed in the next section.

4.3 The Accordion method

4.3.1 Aggregation of Time Series

Aggregate functions provide a means for summarizing a series of measurements in a window, in various ways, as illustrated in Figure 4.2. Different aggregate functions capture different aspects of the measurements within a window. Although the space of aggregate functions is conceivably very large, we have opted for a relatively small collection of functions that represent common statistics of sets of values. The set of aggregate functions, A, considered in this paper is composed of:

- *avg*: the mean value,
- *med*: the median,
- *max*: the maximum value,
- *min*: the minimum value,

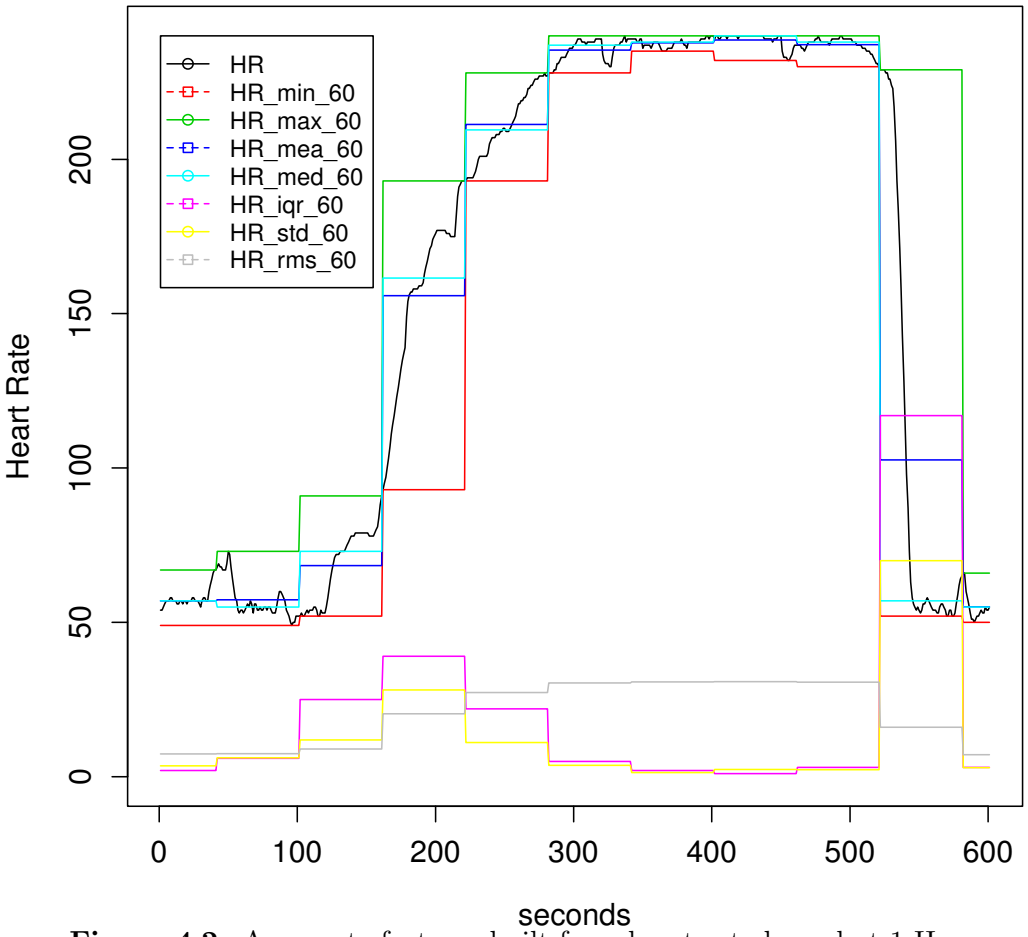

Figure 4.2: Aggregate features built from heart rate logged at 1 Hz.

- *stdv*: the standard deviation,
- the inter-quartile range: $IQR = inf\{x \in \mathbb{R} : 0.75 \leq P(X \leq x)\} - inf\{x \in \mathbb{R} : 0.25 \leq P(X \leq x)\}$,
- the root mean squared: $RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$.

One could argue that features of windows from the *frequency domain*, such as properties of the spectrum of the data, could also be interpreted as aggregate functions: they take a set of measurements (in fact, a sequence), and summarize them into a single value. Such features would capture more periodic properties of the data in the window. Because our algorithms are open source we argue the set of aggregate functions, A, should be seen as mutable. Domain knowledge could be used to change it. If inclined to do so, one can change (add and remove) the aggregate functions that compose A, in order to capture cyclical aspects or peculiarities of the data. The remainder of this paper assumes always the same rather small list of statistical aggregate functions, but there are no technical reasons why other aggregate functions could not be involved also.

Algorithm 4 CalculateAF

Input: time series $\mathbf{s}$, aggregate function $a \in A$, window size w, ratio $q = f_{\mathbf{s}}/f_{\mathbf{r}}$.
  for $i' \in \{1, \ldots, |\mathbf{r}|\}$ **do**
    $i = i' \cdot q$
    if $i < w$ **then**
      $\mathbf{af}_{\mathbf{s},a,w}[i'] \leftarrow a(\mathbf{s}[1, \ldots, i])$
    else
      $\mathbf{af}_{\mathbf{s},a,w}[i'] \leftarrow a(\mathbf{s}[i - w + 1, \ldots, i])$
    end if
  end for
  return an aggregated feature: $\mathbf{af}_{\mathbf{s},a,w}$

An aggregate feature results from the use of an aggregate function a, applied to a *predictor* $\mathbf{s}$ using a sliding window with length w, as formally described in Algorithm 4. To construct an aggregate feature, Algorithm 4 slides a window over the predictor using the reference indices $i \in \{q, 2q, \ldots, |\mathbf{r}| \cdot q\}$. For each reference index i, the algorithm checks for boundary limitations.

If i is smaller than the window size w, the aggregate feature's i'th instance takes into account only the available data from the predictor, $\mathbf{s}[1, \ldots, i]$. Otherwise, the window is $\mathbf{s}[i - w + 1, \ldots, i]$. The aggregate function is then applied to the data in the window.

4.3.2 Feature Construction

This section describes the construction process of multiple aggregate features, as a search problem for the optimal combinations of aggregate function a, high frequency predictor $\mathbf{s}$, and window size w. The objective of the feature construction process is both to slow down the sampling frequency of one or more predictors, and to transform these into good aggregate features. A good aggregate feature should properly describe a target variable ($\mathbf{r}$) at its low sampling rate, $f_{\mathbf{r}}$.

In order to avoid brute force feature construction, and direct the search towards an optimal choice of $(\mathbf{s}, a, w)$, a ranking measure ranking the feature candidates is required. The ranking measures are obtained by the use of scoring functions, $SC(\mathbf{r}, \mathbf{af}_{\mathbf{s},a,w})$. To deal with classification problems, we considered the well-known entropy-based scoring function *information gain* (see for example [80]). For regression problems, the *cross-correlation* [63] scoring function is selected:

$$\gamma_{\mathbf{r},\mathbf{f}}(\tau) = E[(\mathbf{r}_i - \mu_{\mathbf{r}}) \cdot (\mathbf{f}_{i-\tau} - \mu_{\mathbf{f}})],$$

where τ is the time lag between an aggregate feature $\mathbf{f}$ and a target variable $\mathbf{r}$. In the presence of a delayed relation between action ($\mathbf{f}$) and reaction ($\mathbf{r}$), cross-correlation allows the identification and construction lag regression models [31].

The process of feature construction is detailed in Algorithm 5. This algorithm performs a grid search over the available predictor time series in $\mathbf{S}$ and the aggregate functions in A (the two outer loops). The number of different values for both these parameters of an aggregate feature is generally limited, so all combinations will be considered exhaustively.

For each choice of $\mathbf{s}$ and a, the algorithm returns the best window size w_{best}. In essence, this is a task of linear optimization of an unknown function for a given parameter w. In order to avoid a simple exhaustive search for the optimal window size, we sample this function iteratively, and zoom in on a promising interval $[w_l, w_h]$ at each iteration. This heuristic optimization algorithm works as follows.

Algorithm 5 ConstructCandidates

Input: set of predictor time series $\mathbf{S}$, target variable $\mathbf{t}$, scoring function SC, decision *threshold*, maximum window growth ω, number of steps m.

$\mathcal{C} \leftarrow \varnothing$
$q \leftarrow |\mathbf{S}|/|\mathbf{r}|$
for all $\mathbf{s} \in \mathbf{S}$ **do**
  for all $a \in A$ **do**
    $\lambda \leftarrow 1$
    $w_{best} \leftarrow 0, score_{best} \leftarrow 0$
    $w_l \leftarrow q, w_h \leftarrow q \cdot \omega$
    $stop \leftarrow$ **false**
    repeat
      $\delta \leftarrow \frac{w_h - w_l}{m}$
      $w \leftarrow 0, score \leftarrow 0$
      for all $i \in \{1, \ldots, m\}$ **do**
        $\mathbf{f} \leftarrow \text{CalculateAF}(\mathbf{s}, a, w_l + i \cdot \delta, q)$
        if $SC(\mathbf{f}, \mathbf{t}) \geq score$ **then**
          $w \leftarrow w_l + i \cdot \delta$
          $score \leftarrow SC(\mathbf{f}, \mathbf{t})$
        end if
      end for
      if $score_{best} > score$ **then**
        $stop \leftarrow$ **true**
      else
        $score_{best} \leftarrow score, w_{best} \leftarrow w$
      end if
      $w_l \leftarrow w_{best} - q/\lambda, w_h \leftarrow w_{best} + q/\lambda$
      $\lambda \leftarrow \lambda + 1$
    until *stop*
    if $score_{best} > threshold$ **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{af}_{\mathbf{s},a,w_{best}} \leftarrow \text{CalculateAF}(\mathbf{s}, a, w_{best}, q)\}$
    end if
  end for
end for
return $\mathcal{C}$

The interval is initialized based on a parameter ω which indicates the largest acceptable window growth relative to q (the natural window size). At each iteration of the repeat-until loop, a uniform sample is made of the current interval, in m steps (specified by the user, but typically low). Each candidate window size is used to compute a candidate aggregate function $f = \text{CalculateAF}(\mathbf{s}, a, w_l + i \cdot \delta, q)$ and its associated *score*. By the end of each iteration, the current best window size w_{best} is known, and the interval of inspected sizes is reduced to $[w_{best} - q/\lambda, w_{best} + q/\lambda]$, narrowing constantly the original interval through the iterations, around the current w_{best}. This repeated zooming in on the best window size is continued until a more detailed inspection does not yield a better result.

Both Algorithm 4 and 5 are built under the assumption that both target and predictors are sampled at constant sampling rates, but this assumption can be broken in multiple ways. First consider the situation of a target measured at an unstable sampling rate. To overcome this limitation we would need to recalculate q by replacing $|\mathbf{S}|$ with the median sampling rate of $\mathbf{S}$, and recalculate the reference indexes i to synchronize both target and predictors. Secondly the predictors could be measured at an unstable sampling rate. This would require changing the algorithms from index-based windows into time-based windows. The indexes i' and reference indexes i in Algorithm 4 need to be referenced using timestamps, and the window size w needs to be expressed in time units.

Although exhaustive search over window sizes guarantees finding the absolute maximum, the computational costs of this approach would be unacceptable. For this reason, our algorithm employs heuristic optimization to find the optimal window size efficiently. As the score of aggregate functions is generally well-behaved, this heuristic algorithm will typically find the global optimum, rather than a local one. As an example, Figure 4.3 shows an exhaustive scoring landscape for different window sizes, given a low frequency (days) numeric target, a high frequency predictor (1 Hz) and an aggregate function (RMS). The vertical line represents the best window size w_{best}, as determined by Algorithm 5 in a fraction of the time taken by the exhaustive search. Note how this graph also provides a good example of how the optimal window size may differ substantially from the naive choice represented here by 1 on the horizontal scale.

Each combination of $\mathbf{s}$ and a results in a single w_{best}. The associated aggregate feature is then added to the result set of candidate features $\mathcal{C}$, under the condition that its score is higher than a certain *threshold*.

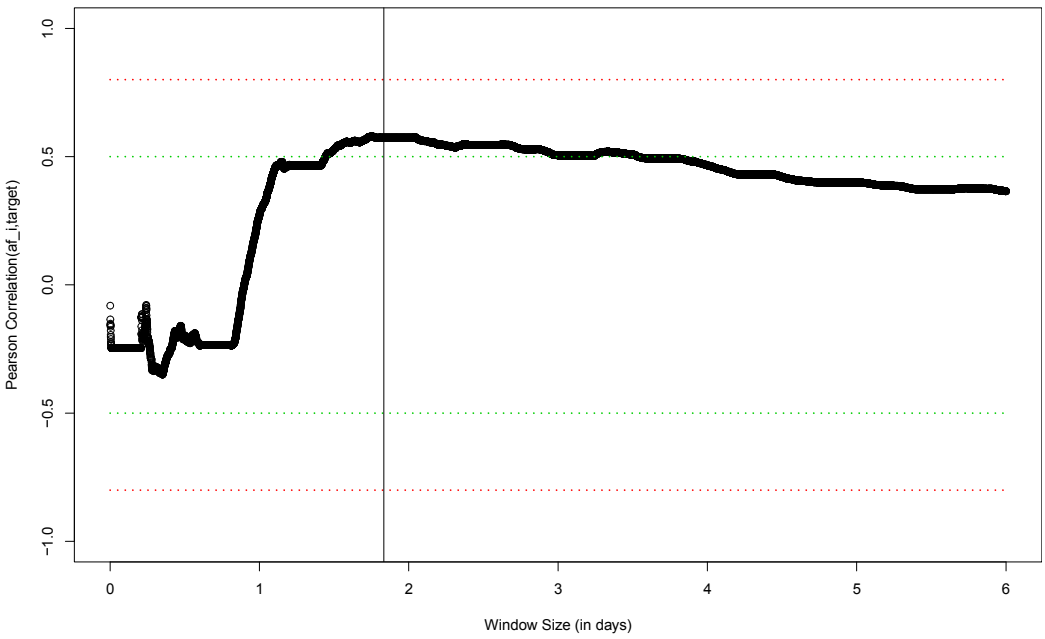

Figure 4.3: Landscape of scoring measure for one of the regression datasets. The horizontal scale indicates multiples of q (in this case 24 hours), the vertical scale the correlation with the target. Note how the optimal window size is almost twice (44 hours) that of the naive choice of $w = q$.

Throughout our experiments, the *threshold* value was set to 0.5 for information gain, and 0.2 for cross-correlation. For ease of implementation, our algorithms will always assume the number of steps m to be equal to the maximum window growth divided into q steps, ω/q, thus effectively removing one parameter.

An important characteristic of our algorithm is that it is memory-conscious. This sets it apart from other feature construction methods [6, 25], because it does not simultaneously materialize most of the inspected features. Keeping only the scores $SC(\mathbf{t}, \mathbf{af}_{\mathbf{s},a,w})$ gives us concise information about how good an aggregate feature could perform, relatively to a target variable $\mathbf{t}$. Every time Algorithm 5 is called, the end result is a collection of at most $p \cdot |A|$ candidate features, which is an acceptable number in most cases.

4.3.3 Feature Selection: Embedded approach

In order to increase the chances of finding dependencies between the aggregate features and the response, we developed an embedded feature selection

Algorithm 6 Aggregate Features Selection: Regression

Input: set of predictor time series $\mathbf{S}$, response variable $\mathbf{r}$, scoring function SC, maximum window growth ω, number of steps m.

$\mathcal{F} \leftarrow \varnothing$
$\mathbf{t} \leftarrow \mathbf{r}$
while $\neg whiteNoise(\mathbf{t})$ **do**
  $\mathcal{C} \leftarrow$ConstructCandidates$(\mathbf{S}, \mathbf{t}, SC, \omega, m)$
  if $\mathcal{C} = \varnothing$ **then**
    return $\mathcal{F}$
  end if
  $\mathbf{f}_{best} \leftarrow \arg\max_{\mathbf{f} \in \mathcal{C}} SC(f, \mathbf{t})$
  $\mathcal{F} \leftarrow \mathcal{F} \cup \{\mathbf{f}_{best}\}$
  $l \leftarrow fitLM(\mathcal{F}, \mathbf{r})$
  $\mathbf{t} \leftarrow \mathbf{r} - l(\mathcal{F})$
end while
return $\mathcal{F}$

method, such that at each iteration of the final modeling algorithm, we do not work with a static set of features. At each iteration, Accordion performs a new feature construction step, and searches for the best aggregate feature.

Regression problems

As the search space of candidate aggregate features can grow too large to explore exhaustively, we resort to a heuristic search that only constructs a subset of promising candidate features, $\mathcal{C}$. When the set $\mathcal{C}$ is large (> 40), a wrapper-based search for the optimal subset becomes impractical [33]. As for backward-stepwise selection in linear models, if the number of candidate features is larger than the number of instances, the use of the least squares method for coefficient estimation becomes impossible [84]. To overcome these potential problems, we employed a *forward-stepwise selection* process [39]. As described in Algorithm 6, at each iteration, we add a new aggregate feature creating a nested sequence of models, until one of the following stopping criteria is satisfied:

- The set of candidate aggregate features $\mathcal{C}$ returned by Algorithm 5 is empty.
- The decomposed target variable is considered white noise.

Algorithm 7 BuildAFTree

Input: set of predictor time series $\mathbf{S}$, nominal response variable $\mathbf{r}$, maximum window growth ω, number of steps m, minimum leaf size $minsup$.

$\mathcal{C} \leftarrow$ ConstructCandidates$(\mathbf{S}, \mathbf{r}, IG, \omega, m)$
if $\mathcal{C} = \varnothing$ **or** $|\mathbf{r}| < minsup$ **then**
  return $\varnothing$
end if
$\mathbf{f}_{best} \leftarrow \arg\max_{\mathbf{f} \in \mathcal{C}} IG(\mathbf{f}, \mathbf{r})$
$c \leftarrow findSplit(\mathbf{f}_{best}, \mathbf{r})$
$\mathbf{r}_l, \mathbf{r}_r \leftarrow \{r \in \mathbf{r} | c(r)\}, \{r \in \mathbf{r} | \neg c(r)\}$
$\mathcal{F}_l \leftarrow$ BuildAFTree$(\mathbf{S}, \mathbf{r}_l, \omega, m, minsup)$
$\mathcal{F}_r \leftarrow$ BuildAFTree$(\mathbf{S}, \mathbf{r}_r, \omega, m, minsup)$
return $\mathcal{F}_l \cup \mathcal{F}_r \cup \{\mathbf{f}_{best}\}$

At each iteration of feature selection, new candidate aggregate features are generated according to the current approximation of the target. In the first iteration, Algorithm 6 uses the response $\mathbf{r}$ as a target to build a set of candidate aggregate features. From this set, it chooses the one with the highest score to add to the set of proposed features $\mathcal{F}$. A linear model is then fitted ($fitLM$) to the response variable $\mathbf{r}$, using the set of proposed features $\mathcal{F}$. In the following iterations, the residual (the part of the signal that cannot be predicted by the current model) becomes the new target variable, $\mathbf{t}$. At the end of the process, only a small subset of the constructed aggregate features is returned.

Classification problems

Decision trees are among the most popular classification models in machine learning, and one of its best-known characteristics is the ability to deal with multiple types of data [80], including trend-less time series. Growing a decision tree involves a divide-and-conquer strategy where each node splits the data into subsets according to conditions on the predictors, until splitting no longer increases the separation between classes. To explore time series with mixed sampling rates, we designed an embedded feature construction and selection method for decision trees, where at each split new features are constructed such that the scoring function information gain (IG) is maximized.

In our method, trees are built recursively. Algorithm 7 shows that at each iteration, a new set $\mathcal{C}$ of candidate aggregate features is constructed, through a search process looking for the best combinations of aggregate features and response variable $\mathbf{r}$. From $\mathcal{C}$, only the aggregate feature that maximizes IG will be used to produce a split: $findSplit(\mathbf{f}_{best}, \mathbf{r})$. The split is then used to create two branches, corresponding to the decomposition of the response variable $\mathbf{r}$, into two subsets $\mathbf{r}_l$ and $\mathbf{r}_r$. The subsets are then used recursively to create more splits until one of the following stopping criteria is satisfied:

- The set of candidate aggregate features $\mathcal{C}$, is returned empty from Algorithm 5.
- The target subset ($\mathbf{r}_l$ or $\mathbf{r}_r$) is smaller than a minimum support, $minsup \in \mathbb{N}^+$.

Note that we specifically do not use pruning techniques during feature construction, to avoid getting too small a feature set. Decisions about pruning strategies can be applied during the final stage of model building by the tree induction method of choice.

4.4 Experiments

In this section, we test our method experimentally, presenting results on the raw data of three datasets collected using multiple sensor systems. The first dataset features a classification problem, involving snowboarding in the Alps. The second and third dataset involve several regression problems, one related to the running speed estimation of an athlete as captured by a GPS sensor, and one describing the amount of sleep of different kinds, as a function of a person's daily routines. The algorithms described in the previous section and further data mining techniques described in this section were implemented in R [81].

In each experiment, we not only compute the results for our embedded method, but also consider traditional two-step alternatives: construction and then selection. Feature construction alternatives include a baseline aggregation method, and grid search feature construction. The baseline aggregates over a non-overlapping window of size $f_{\mathbf{S}}/f_{\mathbf{r}}$ by simple averaging. For the grid search approach, we materialize a rather large amount of aggregate features. The grid search is bounded, such that it generates an aggregate feature matrix of approximately 5 million cells, allowing an absolute comparison across datasets.

Dataset	Target	# Input samples	# Output samples	Method	# Features constructed	# Features selected	time (s)	Accuracy/ R^2
Snowboard	Activity	21 180	353	Accordion	30 180	15	305	**84.7%**
				Baseline	25	10	0.5	67.1%
				Grid search	17 150	15	526	83.1%
Speed Estimation	Speed (m/s)	951 200	2088	Accordion	11 580	2	2 654	**0.986**
				Baseline	6	3	1.99	0.388
				Grid search	2 394	25	1 141	0.508
Daily Routines	Light (h)	1 296 575	15	Accordion	19 260	4	2 334	**0.8407**
				Baseline	34	6	28.7	0.2089
				Grid search	323 400	36	62 409	0.8391
	REM (h)	1 296 575	15	Accordion	23 610	5	3 105	**0.9184**
				Baseline	34	4	28.7	0.1837
				Grid search	323 400	11	62 409	0.8838
	Deep (h)	1 296 575	15	Accordion	11 710	2	1 451	0.6719
				Baseline	34	4	28.7	0.0842
				Grid search	323 400	15	62 409	**0.8457**

Table 4.1: Experimental setup and results.

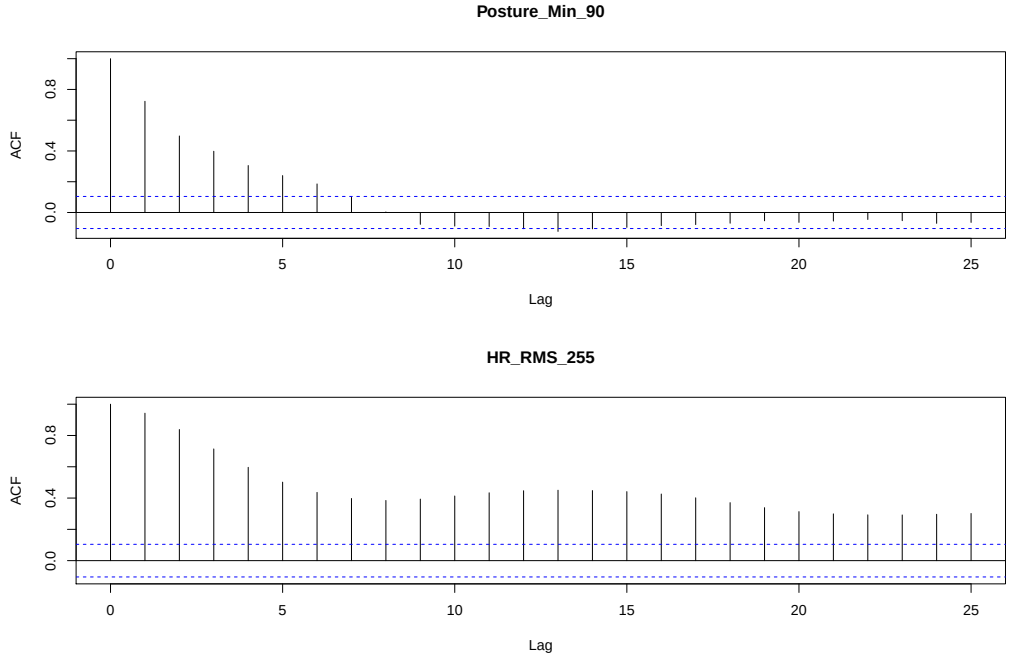

Figure 4.4: The plot shows the autocorrelation behaviour for two randomly sampled aggregate features, from the Routines dataset.

The alternative feature construction methods are followed by well-known feature selection methods, specifically Lasso for regression [89] and C4.5 for classification. For Lasso, we used the default options proposed in the glmnet [26] package in R. To choose the penalty parameter, we employed cross-validation on the training set, and chose the penalty parameter that minimizes the mean squared error.

In most cases, time series have a natural temporal structure. This prohibits us from assuming that subsequent aggregate feature instances are *iid* (independent and identically distributed). This follows from the fact that closer observations have a stronger relation than those further apart. Figure 4.4 shows the autocorrelations of two aggregate features sampled randomly from Accordion's candidate features, $\mathcal{F}$. As expected, all of them have a clear temporal structure. Breaking the *iid* assumption restricts the model evaluation methods that can be applied. For example, cross-validation should only be applied when features can be assumed to be *iid*. Consequently, we split the data into 66% of data before a selected point in time (the training set), and 34% for testing.

4.4.1 Snowboard Data

This experiment involves sensors collecting physiological signals from a subject while doing winter sports in the Alps. To collect this physiological data, we used a Zephyr BioHarness 3[2] sensor system, that is worn on the subject's chest. The BioHarness incorporates multiple sensors (ECG, chest expansion, temperature and tri-axial acceleration), that are embedded in a monitoring module and a lightweight strap. The system samples at multiple sampling rates for the different sensors, and derives from them a total of 25 physiological parameters (heart rate, breath rate, posture, peak acceleration, ...), logged at 1 Hz. The dataset used in this experiment was collected during 353 minutes of snowboarding. During the collection period, the subject used the BioHarness and a GoPro Hero3 HD camera. Afterwards, the video data was used to label the activities for each minute, from the following available labels: *lift, lying, sitting, snowboarding, standing* and *towlift*.

The baseline consists of 25 averaged predictors, with a sliding window of size 60, at 1/60 Hz (once per minute), matching the frequency of the target labels. Table 4.1 presents information for each decision tree built, where the baseline achieved a predictive accuracy of 67.1%. We used an implementation of C4.5 from the package rWeka[3], which allows R users to use Weka's[4] machine learning algorithms. Reduced error pruning was used as post-pruning strategy.

We employed the classification version of Accordion (Algorithm 7). The algorithm used information gain as scoring function, and was allowed to grow windows up to 5 minutes in length. After considering 30 180 candidate aggregate features, only 15 were selected to compose the set of aggregate features ($\mathcal{F}$). Figure 4.5 presents the resulting decision tree, with a prediction accuracy of 84.7% on the test set. At the root of the tree, active activities are separated from passive ones based on the minimum heart rate over the last minute. The right side of the tree distinguishes between active or recently active activities using heart rate, acceleration and breathing as input predictors. The left side of the tree predominantly uses acceleration variables to classify between the different passive activities. Since the predictors were logged at 1 Hz, the window sizes can be interpreted as the number of seconds aggregated. The variety of window sizes and aggregate functions (see Figure 4.5) reveal features with multiple degrees of integration over time.

[2]http://www.zephyranywhere.com/products/bioharness-3/
[3]http://cran.r-project.org/web/packages/RWeka/index.html
[4]http://www.cs.waikato.ac.nz/ml/weka/

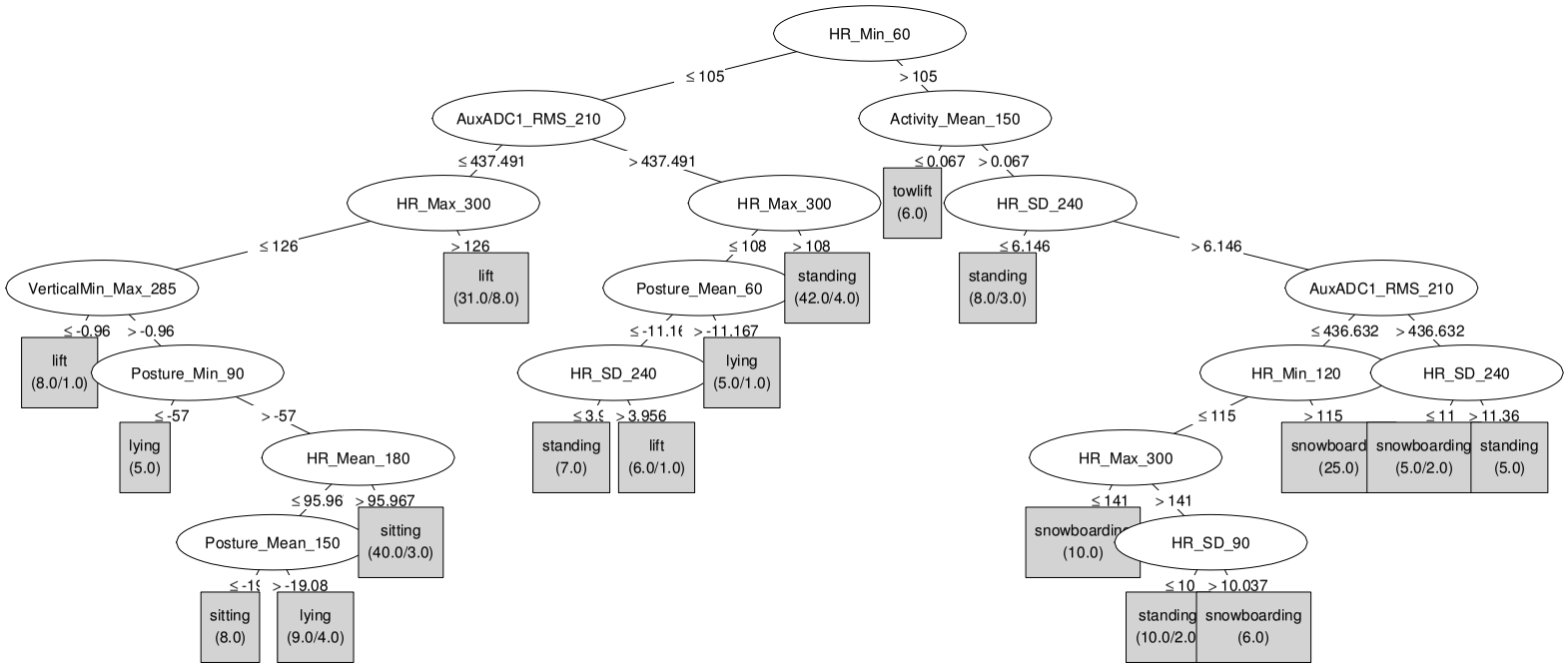

Figure 4.5: Decision Tree C4.5 implemented in Weka, built with the features proposed by Accordion.

The grid search approach materialized 17 150 aggregate features and fed it to C4.5 to build a model from a subset of these. The number of features constructed corresponds to a matrix of 5 million cells, as described before. This involved trying 80 different window sizes for each pair of aggregate function and predictor. Table 4.1 shows that Accordion outperforms grid search both in computation time and model accuracy.

4.4.2 Speed Estimation

This dataset was collected in the context of an athlete training for the Amsterdam marathon. In this context, two accelerometers[5] were worn by the athlete during four training sessions, one strapped to the right wrist and the other to the right ankle. A Garmin Forerunner[6] device was used to measure distance and speed.

The dataset considered here has as input about 2 hours and 40 minutes of running measurements from 2 triaxial accelerometers ($2 \times 3 \times 951\,200$ data points), at a constant sampling rate of 100 Hz. For the same measurement period, the target speed values were extracted from the Garmin Forerunner GPS, of which the median sampling rate is 0.2 Hz. The speed (in m/s) turned out to be captured at an unstable rate, with time lapses between measurements ranging from 1 to 10 seconds. Having an unstable target sampling rate is a specific challenge of this dataset, but one that can fairly easily be handled by our algorithms.

The design of our algorithms assumes that all the measurements are done at constant sampling rates. Note that, as long as the predictors are collected at a constant sampling rate, having an unstable target sampling rate (as is the case here) is not a problem. We used this specific challenge to show how our algorithms can be made to work on a broader set of tasks. In fact only two changes are needed. First, in Algorithm 5, recalculate the relation between predictors and target sampling rates to $q = 100/0.2 = 500$, to reflect the median sampling rate of the target. Second, as a minor modification of Algorithm 4, we calculate beforehand the reference indices i, such that predictors and target variable can be synchronized properly.

As baseline experiment, we aggregated 6 variables (2 accelerometers $\times$ 3 axes) over non-overlapping windows of variable sizes. With the predictors

[5] http://www.geneactiv.co.uk/

[6] https://buy.garmin.com/en-US/US/into-sports/running/cIntoSports-cRunning-p1.html

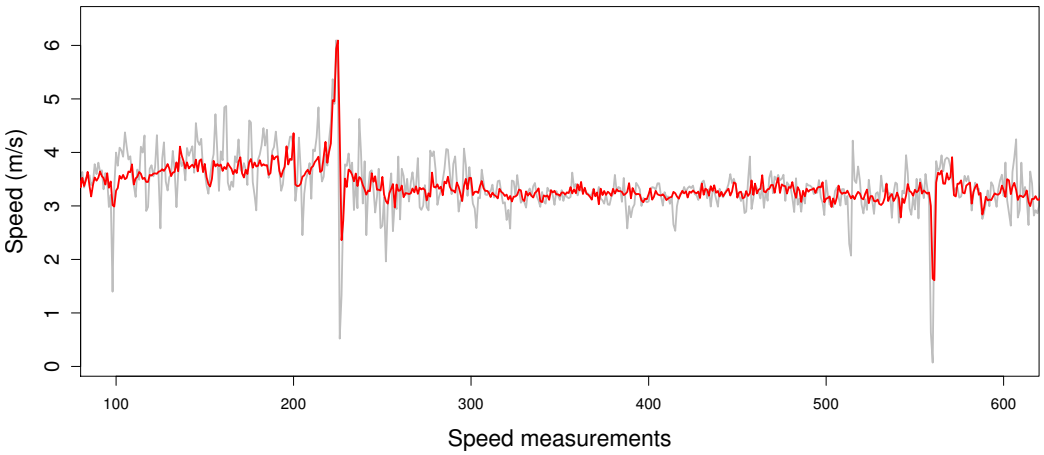

Figure 4.6: Predicted value (black) vs. real speed (gray).

and target variable synchronized, the Lasso regression selected only 3 features, with a fairly low coefficient of determination ($R^2 = 0.388$). As for grid search, using the limit of creating a feature dataset bounded to 5 million cells, we materialized 2 394 aggregate features in about 1 141 seconds, and subsequently submitted them also to Lasso. The achieved $R^2 = 0.508$, although higher than baseline, could not outperform Accordion ($R^2 = 0.986$).

The scoring function chosen in this experiment was cross-correlation, enabling so-called *lag regression*, and the maximum window size was set to 60 seconds. During the iterative process of construction and selection, 11 580 aggregate features were constructed, from which only two were selected (thus, two iterations). The resulting predictions on hold-out data are shown in Figure 4.6. The final lag regression model is as follow:

$$\begin{aligned} speed[i'] &= 1.198 \cdot af_{\mathbf{ankleY},RMS,200}[i'] \\ &\quad +0.495 \cdot af_{\mathbf{ankleZ},stdv,5886}[i'-4] + e[i'] \end{aligned}$$

Interestingly, the accelerometer strapped to the wrist was never selected to model the speed of the athlete. Also, the selected features use both short term (window size of 200 equals 2 seconds of accelerometer data) and long term information (window size of 5886 aggregate approximately 1 minute). Both these observations can help domain experts redefining future data collections and understanding multiple temporal phenomena. All this insight can be leveraged by the usage of aggregation functions well known to specific domains.

4.4.3 Daily Routines Data

Subsequently, we tested our method on another dataset involving three regression targets, related to the amount of time spent in *light sleep, REM* and *deep sleep*. During the course of 15 days, a subject produced data in the context of a self-tracking experiment, collected using various sensoring systems: a) the Zephyr BioHarness (see above), used during the day (except during bathing), b) OpenBeacon, an RFID wireless sensor system to monitor the time spent at different locations of the home, and c) a Beddit[7] sleep monitoring system to monitor the nights. This last system is used both for recording the breath and heart rate during the night, as well as determining the different sleep stages at night (a computation that is part of the black-box service of Beddit).

The dataset used for this experiment consisted of 34 input attributes, sampled at 1 Hz, of which 24 are physiological variables from both the BioHarness, and Beddit. The remaining 10 variables are binary. They refer to the subject's location and were extracted both from the OpenBeacon and Beddit sensor systems. Table 4.1 summarizes the experimental setup for all targets, as well as the information about results.

As baseline experiment, we used the same idea of feature construction for the previous baselines. Although the baseline is quite fast in terms of computation, the R^2 results (on the test dataset) show that it is a naive solution. As for the grid search solution, we materialized 323 400 aggregate features after about 17 hours of computation, which is considerably slower than what Accordion took to construct and select its aggregate features. After performing Lasso to all the targets, the R^2 results show that this alternative was outperformed by Accordion in two of the three targets.

For Accordion, we used cross-correlation as a scoring function, and the maximum window size allowed was 3 days. With our method, the target that took the longest to compute (about 50 minutes) was the REM stage of sleep. REM sleep is considered the lightest stage of sleep [36]. Figure 4.7 shows the stack of nested models created by forward selection. For this target, the algorithm constructed 23 610 aggregated features, from which only 5 (5 iterations) are proposed to explain REM linearly.

[7] Available from http://beddit.com

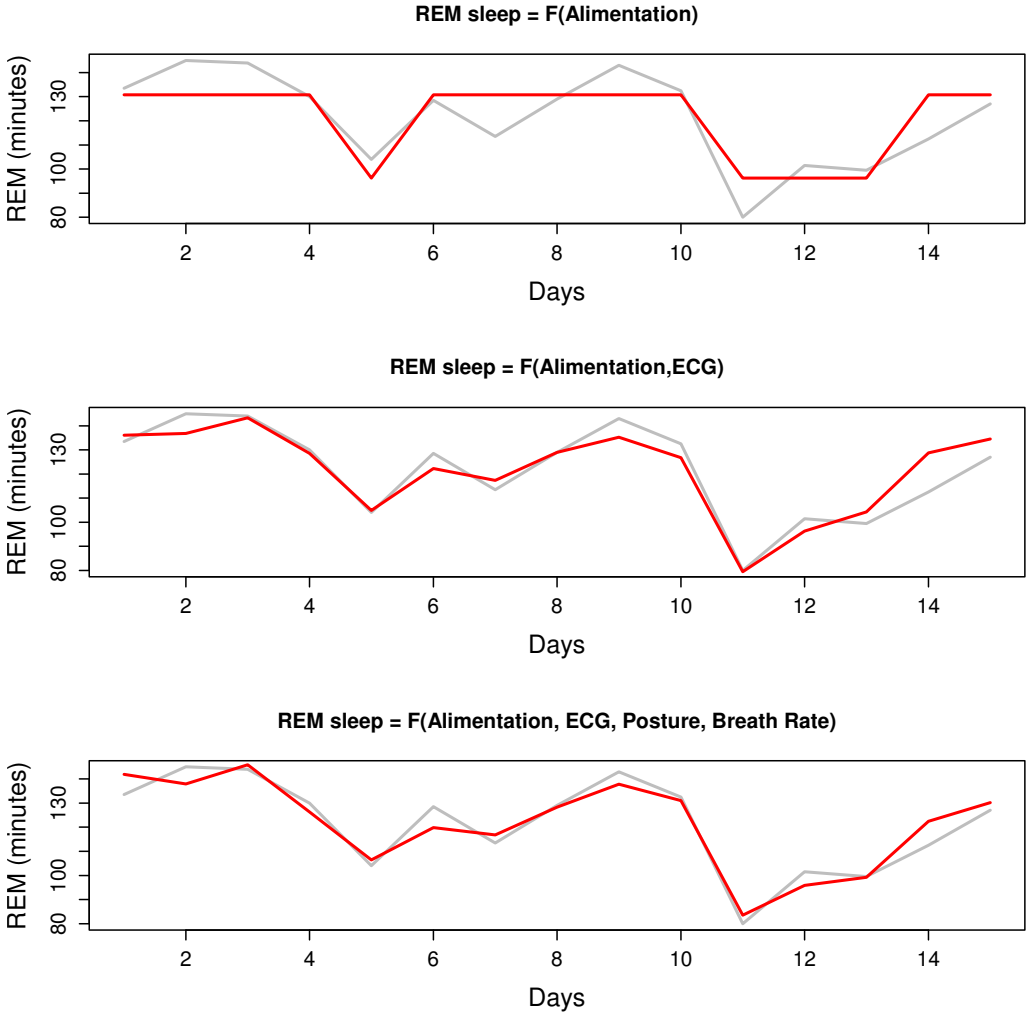

Figure 4.7: Predicted value (black) vs. real amount of REM sleep (gray), for models based on the first single, two and five (all) features, respectively.

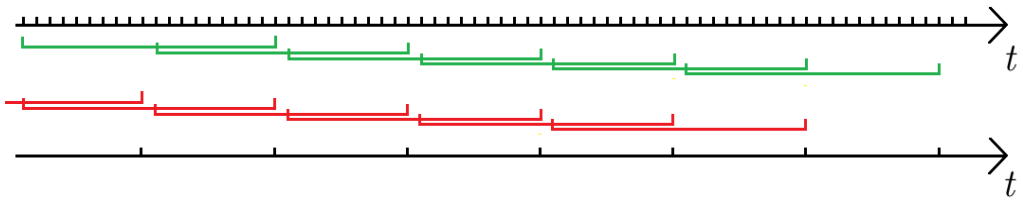

Figure 4.8: High frequency time series can be transformed into aggregated features, resulting in a linear lag regression between these and the target variable.

From the three targets addressed in this experiment, time spent in deep sleep gives a good example of both failure and success of our method. Accordion model scored lower than the one produced Lasso with grid search in terms of R^2, where Lasso wins over our *forward-stepwise selection* process in terms of model accuracy. On the other hand, Accordion is faster and produced an interpretable model (two selected aggregate features), whereas Lasso with grid search fails to deliver an interpretable model (check Table 4.1: R^2 and #Features selected). Our method produced 11 710 candidate features, from which only two were selected to explain the amount of deep sleep, resulting in the following linear model:

$$\begin{aligned} deepSleep[i'] &= -120.8 + 0.99 \cdot af_{\mathbf{Posture},RMS,158400}[i'] \\ &\quad +0.11 \cdot af_{\mathbf{HR},avg,155200}[i'-1] + e[i'] \end{aligned}$$

Figure 4.8 helps us to interpret the *deepSleep* model. Both aggregate features have window sizes of about two days, with almost 50% of overlapping.
For each moment i', deep sleep can be explained with posture over the last two days $af_{Posture,\ RMS,158400}(i')$, and the heart rate of almost two days with a delay of one day $af_{HR,\ avg,155200}(i'-1)$.

4.5 Related Work

The problem of activity recognition is commonly tackled with a two-stage process [6, 69, 58]: first, manually construct aggregate features and then apply a machine learning technique to discriminate between different activities. The task of feature construction is so central that surveys of feature construction techniques became necessary [25]. As the choices in feature construction influence all the experiments, this often leads to solutions that are overly specific to the experimental setup (sensors used, data collection, application). As a result, most methods are not generic [12]. Our algorithm embeds feature construction into the learning process, which increases the feature search space, reduces the time spent pre-processing data and avoids overly specific solutions, which makes it widely applicable.

From a data mining perspective, the use of time series as a data source has received considerable attention, and has developed into different areas of research [48, 102, 60], e.g. classification, summarization, subsequence clustering, motif discovery and anomaly detection. As for the challenge of mining time series with mixed sampling rates, this still remains underappreciated.

To the best of our knowledge, this paper is the first attempt to develop a data-driven generic solution to this problem, and to focus on the importance of optimizing the *automatic and dymanic construction and selection* of aggregate features with respect to a target variable, as opposed to static feature construction.

In econometrics, regression models are commonly used to relate variables at the same sampling frequency, even when the data sources are being collected at different rates. When dealing with mixed sampling rates, the most common technique is still to downsample the predictors [4], or upsample the target variable [3]. Recent work proposes solutions to forecast directly from variables with mixed sampling rates, both for univariate [28] and multivariate time series [57]. These proposed methods still rely mostly on the expertise and creativity of the economists (domain knowledge-driven), leaving no room for data-driven knowledge discovery, which our algorithm is capable of doing.

4.6 Conclusions and Future Work

When modeling time series for activity recognition, a drawback is the considerable amount of time required to pre-process them into good features, a process that often calls on domain knowledge about the underlying problem. Accordion shortens the pre-processing time, generating candidate aggregate features automatically by optimization of its components, $(\mathbf{s}, a, w)$. We still believe that domain knowledge can play a big role when modeling, but this effort could be redirected to higher level questions, such as which set of aggregate functions (A) to use. One of the directions for future work is naturally to extend the set of aggregate functions, especially to the frequency domain.

We also motivate the idea of embedding automatic feature construction into the machine learning process. The idea here is to stop relying on static sets of features, and at each iteration of feature selection direct the search for good candidate features. Making use of scoring functions, Accordion is able to test many candidate features, and return only a small set of selected features. Especially when quick but reliable results are required, or large datasets dictate a memory-conscious method, our algorithm is clearly a good choice. As future work, we would like to mimic the learning process of other supervized methods, both in regression and classification, keeping the idea of learning algorithms that do not rely on a static set of features.

One of the achievements of our approach is that it outputs interpretable aggregate features. The ability to interpret our aggregate features follows from the combinations of input variables, well-known aggregate functions and different window sizes. Interpretation of different window sizes come from the fact that our method searches for different phenomena by expanding or contracting the window size for each feature. In contrast, the standard approach in activity recognition is to take a more or less arbitrary choice about a window sizes [6, 69, 58, 37]. In the future, we would like to deal with unstable sampling rates, both of predictors and target, multiple sampling rates for the predictors, and targets at a *higher* sampling rate than the predictors.