CHAPTER-4
PROCESSING WIRELESS SENSOR DATA

4.0 CHAPTER INTRODUCTION

Stage 1 of the proposed methodology has been studied and implemented in this chapter. Data streams produced from sensors is redundant and noisy. It can also be in varied forms. Simple sensors produce numeric data while some complex sensors like camera and microphones emit data in the form of video, images and audio. Some switch based sensors output binary or categorical valued data also. Transmission and use of sensor data streams in its raw form is neither cost effective nor useful. Data is thus processed before transmission to next device, with the objective to remove its imperfections, decrease cost of transmission, decrease storage requirements and enhance usability.

Any single method cannot fulfill all objectives and multiple steps with specific techniques have to be used to address each problem. These steps have been described in this chapter and are proposed to be applied at each sensor node to process locally collected data before transmission. Given the resource constraints of sensors, the processing methods need to be light weight in terms of computation and storage requirements [164]. This chapter deals with study, selection or design of such processing techniques that can ensure the quality of processed data. The chapter has been organized as follows.

In first section, framework for sensor data processing is described. Justification and techniques of different processing steps like smoothing, missing value replacement, segmentation, feature extraction and discretization have been discussed. Subsequent sections describe each step and their evaluation in detail.

In Section 4.2, methods of imputing missing values in the stream and smoothing have been discussed. For assessing the effect of imputation, random values have been missed from data and
‘correction error’ has been found for evaluation. Three popularly used light-weight methods have been evaluated for smoothing using normalized root mean square error metric.

In section 4.3, approaches of stream segmentation have been discussed. Various segmentation lengths and overlapping have been evaluated. In Section 4.4, segments have been subjected to outlier identification and replacement. Transformation of segmented sensor data as features has been described and their effect on processed data quality has also been discussed in this section.

Significance of discretization of real valued features to discrete values and some related methods have been discussed in Section 4.5. Effect of processing steps on energy efficiency has been evaluated analytically in Section 4.6. The chapter is concluded in Section 4.7.

4.1 SENSOR DATA PROCESSING FRAMEWORK

It is apparent from analysis of sensor data characteristics in Chapter 2 that absolute sensor values have to be transformed to represent useful semantics of application. It has been evident that raw data can be clustered at intra node level to reduce redundancy without affecting data quality [165]. In this section, framework of various processing steps applied on sensor data and their order of implementation has been described. Consider raw data vector $X_t = \{x_{t1}, x_{t2}, \ldots, x_{tm}\}$ at a time instant $t$, from a set of sensors $S = \{s_1, s_2, \ldots, s_n\}$ where $n \leq m$. The data processing model, $P$, maps the above multidimensional dataset $X_t$ to another multidimensional set, $Y_{t'}$, at another time, $t'$

$$Y_{t'} = \{y_{t'1}, y_{t'2}, \ldots, y_{t'm}\}, \text{ where } \Delta t' > \Delta t$$

Elements of $Y_{t'}$ are lowest level of semantic abstractions from a single sensor and have been termed here as “context attributes”. The time interval of sensor data generation is lesser than time interval for “context attributes” generation due to intra node aggregation before generation.

Proposed processing is done sequentially as shown in Figure 4.1. Treatment of sensor data before transmission to sink will be done in multiple steps serving two purposes viz. “data cleaning” for sanitization and “data transformation” for enhancement of information content.
Processed data set $Y_t$, is considered to be semantically more useful for further interpretations as raw, redundant, continuous and noisy sensor data is transformed into cleaned, compressed and nominal context attributes.

First step in data processing is smoothing of data. Purpose of smoothing is to remove transient random noise and sudden aberrations without affecting the true changes in signal. Smoothing ensures that processing is not swayed by accidental spikes in data. Continuously produced sensor signal may have certain values missing due to uncertainties in electrical circuitry. Missed values make further processing difficult; hence they are to be handled by discarding or online replacement before moving to next step.

Raw sensors data is often sensed at a high rate of 50 – 100 Hz typically. Real life contexts of interest do not vary at such frequency but span over some period of time. An aggregated view of sensor data at a lower rate can thus be obtained. The aggregation interval has been termed here as segmentation window. Studies have been done in this chapter to optimize segment window length and inter- segment overlapping.
Data in each segmented window has been tested online for presence of intra-segment outliers using commonly used simple methods. Detected outliers have been replaced using mechanism similar to missing value replacement. Appropriate features have been derived from sensor data of each segmented window to better represent characteristic of segment than raw data. Simple features like standard deviation, mean, median, mode etc. are suitable for real time extraction. For example, while standing, absolute value of z axis of an accelerometer (worn on arm) would be different for persons of different heights but standard deviation of z axis data can recognize standing.

The next step is conversion of real valued features into discrete valued data. This is important for compact representation and conciseness of specification. Nominal values obtained on discretization also improve understandability. It makes learning of higher level classifiers faster and more accurate [178]. Sometimes, for well understood data, it is straightforward to discretize raw data by cutoff interval values manually. Algorithmic discretization has been applied whenever, this is not possible.

For every step, application specific parameters are to be learnt by training. Sufficient samples of archived data will be used for training. Training data can be stored at sink or in intermediate devices named as “Context Heads”. A Context Head (CH) is distinguishable from sensor in computing power, type of device and capability of movement. Devices with decent computing powers like smart phone and laptops can act as CHs. A sensor system may have several such devices depending upon the physical scale of deployment.

Both sensors and CHs operate in two modes namely, training and context extraction. In training mode, raw sensor data is transmitted to CHs from each sensor. Step specific optimal parameters are then learnt at respective context heads and finally communicated to sensor nodes post training. In context extraction mode, all context heads receive processed data from designated sensors.
System operates in *training* mode at the beginning of operation and enters in *context extraction* mode when time for training is over. Data flow in both modes using parameters learnt from training has been summarized in Figure 4.2.

**Figure 4.2:** Processing by Sensor Nodes in Various Modes
After these steps, raw data of different rate and form is transformed to discrete valued context attributes at uniform rate. Last two steps of processing are lossy transformations and may compromise data quality. Step wise appropriate metrics have been applied to evaluate the effect of each step on quality of data. Indirect validation method of evaluation of accuracy of context classification will be used in next chapter to assess the cumulative effect of these steps. The localized processing computations yield significant energy savings also. Detailed algorithms of each processing step and analytical evaluation of overall energy savings follow in respective sections.

4.2 MISSING VALUE DETECTION AND SMOOTHING

4.2.1 Missing Values Imputation

Values at few instances may go missing at the sensors due to problems like sensor power outage, external interferences etc. Missing values can be either discarded or replaced with some other values. The process of systematically replacing missing values is called “imputation” [166]. Existing approaches for missing imputation in databases are based on utilizing archived data. Methods like maximum likelihood estimation, expectation maximization, regression has been popularly used to estimate imputed values in such databases [167 - 168]. However, these methods are not suitable for sensor data due to its stream nature. A stream is continuously evolving and it’s unreasonable to archive all values or impute according to all archived values, as the stream data may not be related to all past data. An issue with missing values treatment is about which samples should be considered for estimating replacement. The selected samples should be related to missing data. For online imputation, method should be simple enough to compute the imputed value in real time.

In this work, simple method of using last available value has been used for imputation. Given high sampling rate of sensors, consecutive values of same sensor are highly correlated temporally. Thus for simplicity of computation, replacing missing value with last available value of same sensor is considered most appropriate intuitively.
Data imputation for whole duration of operation, $dur_{op}$, using last available value for $m$ dimensional stream sensor dataset can be stated as function $impute()$.

Function $impute()$

begin
  while $(t < dur_{op})$
    for $(i=1; i<m; i++)$
      if ismissing($x_i^t$) then
        $x_i^t = x_{i-1}^t$
      end if
    end for
  end while
end

Thus, the imputation model for missing values at sensors will perform in following manner. It first obtains the raw sensor value and evaluate if value is missing or unreadable. If it is so, data imputation according to above algorithm is done and result is presented to next processing step. Otherwise, raw value is used as it is in next step.

For evaluating the performance of imputation methods, missing values may be deliberately introduced and the performance can be determined as correction error. Using last available imputation, correction error was found to be less than 10% for up to 25% missing values.

4.2.2 Smoothing

In practical sensor based implementations, specifically in human activity recognition, true sensor signal changes smoothly as a function of time. Rapid and random transitions can be considered as noise. In this step, it is attempted to reduce this noise by $Smoothing$ methods.

In smoothing, values of adjacent signals are modified so that each value falls in a greater than or less than range of each other. Such modification leads to a smoother signal. If the actual sensor data is not aberrant, then noise will be reduced without losing original signal’s actual characteristic. Moving Average Models and Exponential Smoothing Models are useful over short time periods [43].

$Simple$ $Moving$ $Average$ $(SMA)$ $/$ $Median$ $(SMM)$ $Method$

In simple moving average, smoothed value of $X_t$, $smoothed(X_t)$ is computed as the average of $2*N$ adjacent points and $X_t$. For each point, sensor data values of ‘N’ points before and ‘N’ points after it are used.
\[ smoothed(X_t) = \frac{1}{2+N+1} \sum_{h=t-N}^{t+N} X_h \] (4.1)

The number, \(2*N+1\) is called smooth span.

In Simple Moving Median (SMM), smoothing process is quite similar to SMA, except that median is utilized as basic mathematical operation over smoothing span instead of average. SMA and SMM are un-weighted smoothing methods in the sense that all the participating values in smoothing are given equal weightage. A weighted smoothing is considered to be more appropriate due to higher weightage being allotted to recent values.

**Exponential Moving Average (EMA) Method**

EMA is smoothing method using an exponential weighting or smoothing factor, \(\alpha\) between 0 and 1. The smoothed signal is then computed as per eq. 4.2.

\[ smoothed(X_t) = \alpha * (X_t) + (1 - \alpha) * smoothed(X_{t-1}) \] (4.2)

The most recent value is assigned a weight of \(\alpha\) and last smoothed value is weighted by \((1 - \alpha)\).

It can be seen that, \(\alpha = 0\) results in any new data having no effect, while taking \(\alpha = 1\), means that no smoothing takes place and data is used as it is. Taking \(\alpha\) somewhere between 0 and 1 can only result in smoothed data. \(\alpha\) has been taken here as \(\frac{2}{\text{span}+1}\).

**Effect of Smoothing**

The process of smoothing was intended to give a visually better signal as compared to the original signal while maintaining the sharp step response. All the three methods described above were applied to **Opportunity Dataset (Refer Appendix A for Details)** to assess this effect. Smoothing was applied on raw data of accelerometer worn by Subject 1 above his right knee while \textit{“walking”}. It can be seen in Figure 4.3 that sudden sharp spikes in y axis data of the subject are flattened without losing the differentiability. A smoothing span of three adjacent values has been utilized to have good smoothing without introducing much lag.
Quantitative effect of smoothing on quality of data has been evaluated as Normalized Root Mean Square Error (NRMSE). NRMSE for smoothing performance has been calculated as per equation 4.3.

$$\text{NRMSE}(D^i) = \frac{\text{RMSE}(D^i)}{D^i_{\text{max}} - D^i_{\text{min}}}$$ (4.3)

where \( \text{RMSE}(D^i) = \sqrt{\frac{\sum_{t=1}^{\text{dur,op}}(\text{smoothed}(d^i_t) - d^i_t)^2}{\text{len}(D^i)}} \) (4.4)

\( D^i \) is a column vector of entire raw data of \( i^{th} \) dimension and \( \text{len}(D^i) \) is the total number of data points that were smoothed.

NRMSE of smoothing right knee accelerometer data of Subject 1 while doing several types of locomotion is shown in Figure 4.4.
It can be seen from Figure 4.4 that on using similar smoothing span, EMA gives least NRMSE. Similar results were obtained for data of other subjects also. Thus, EMA is chosen as the preferred smoothing method for online smoothing of sensor data in this work.

4.3 SENSOR DATA SEGMENTATION

A deployed sensor generates streams of data. Data streams are managed by considering multiple consecutive samples or ‘segments’ as operation window such that each window can be processed at a time. Besides enabling stream processing, sensor data segmentation may also serve following purposes:

- Sensors in HeWSNs sense at variable sampling rate. For example, among wearable sensors, PPG and ECG are sampled at 200Hz, accelerometer at 30 Hz, body temperature typically at 0.2 - 1 Hz and heart rate at 1/60 Hz (1 per minute). Segment size of common duration can be used to make data rate of each sensor similar. An example for this has been shown in Figure 4.5.
- Data at original sampling rate is not of interest to end user and applications directly. Nintendo’s Wii Remote samples accelerometer data at about 100 Hz. These rates are manifold higher than the rate of change of situations that are of interest to humans.
Segment based compression can decrease transmission rate to reduce energy and memory requirement of storing and forwarding raw sensor data.

Segmentation is represented graphically in Figure 4.5 by synthetic sensor datasets, where all three sensors have different data rates and a segmentation window of 5 seconds is chosen. It can be seen that number of samples per segment per sensor will be different.

Philosophically, problem of segmentation is about dynamically deciding appropriate window size such that readings within that window are highly correlated or coherent. Algorithmically, it is about finding most optimal duration of window for online segmentation of sensor data such that the cost of segmentation is minimized and segment quality is maximized [169 - 170].
Formally, Given a temporal sensor stream $D^i$ of $i^{th}$ sensor of the dataset, segment $seg(D^i_{\tau_1:\tau_2})$ has been defined as

$$seg(D^i_{\tau_1:\tau_2}) = \{ d^i_{\tau_1}, d^i_{\tau_1+1}, \ldots, d^i_{\tau_2} \}$$

The length of a segment ($\tau_2 - \tau_1$) is represented by symbol ‘$W$’. As window size is determined in terms of duration, the actual number of samples in a segment depends on the sampling data rate of the sensor.

The cost of segmentation has been defined here as delay introduced due to segmentation. The delay is directly proportional to the segment size. The quality of segment has been defined by as mean autocorrelation of all segments on a given segment size. The autocorrelation function of a segment, $seg(D^i_{\tau_1:\tau_2})$ has been calculated as per eq. 4.5.

$$ACF(seg(D^i_{\tau_1:\tau_2})) = \frac{\sum_{j=\tau_1+1}^{\tau_2} Covariance(d^i_j, d^i_{j-1})}{W} \frac{1}{\text{stddev}^2_{d^i_j} \times \text{stddev}^2_{d^i_{j-1}}},$$

(4.5)

A segment size with larger ACF is considered as more homogeneous segment.

### 4.3.1 Some Segmentation Approaches

Correct sized segmentation of continuous sensor streams is important to obtain distinguishable features. Fixed and dynamic sizes have been proposed in literature [169] [171 - 172]. In fixed size segmentation, each segment is of a pre-defined fixed size, while in dynamic sized windows, segments may be of varying size depending on the incoming data in stream. Use of sliding windows is one of the popular methods of implementation of segmentation. Sensor data segmentation with sliding windows may be done as overlapping sliding or non-overlapping windows, where consecutive time windows share and do not share common data samples respectively.
[169] suggested an online algorithm for creating dynamically sized segments inter-segment Kullback-Leibler distance. A segment boundary was defined when difference in probability distributions of current and historic data became higher than a predefined threshold. A top down approach has been used here and initially two segments of random lengths were considered. These are recursively split into further smaller sequences using criteria defined above. [171] suggested that abrupt change in data can be used as segment boundary. The abrupt change though has not been quantified and any method to separate it from outliers has not been suggested. Systematic iterative top-down and bottom-up segmentation has also been suggested as effective segmentation methods. These require global knowledge of data with ground truth labels and hence can be used only for offline segmentation [170].

A dynamically decided variable size sliding window was suggested in [173]. It suggests merging sensor data of continuous activity as one instance. This is not suitable for long term activity recognition as important varying learning instances of a context attribute class is lost. The method was devised for binary sensors and do not scale well to continuous values being used here. A hybrid approach applying bottom up method with sliding window for achieving improved online segmentation was suggested in [174].

Variable length window for different activities taking into account time decay and mutual information has been proposed. These approaches are supervised and do batch segmentation, therefore are not suitable for online division. Moreover, they may not scale well for multi-valued data.

Fixed length windows are easier to implement, however, there is no exact method to determine best possible window size and amount of overlapping. Few researchers have proposed exhaustive hit and trial to find optimal values for both in underlying application [72][75]. [68] explored activity wise, fixed time sampling approaches to solve the problem of selecting the optimal length of window size for segmenting binary motion sensors. In many other works, an application specific intuitive size has been taken [69 – 70].
Conclusively, there is hardly any simple approach to segment online continuous valued sensor data as optimally sized windows.

### 4.3.2 Proposed Segmentation Approach

An OPtimal Segmentation (OPS) based on target context duration heuristic has been devised here. The window size is decided as the function of duration of actual instances of contexts in training data. It is calculated as the mean of minimum continuous duration of any context labeled in training set. Detailed method is given in algorithm OPS.

/*Optimal Heuristic Computation - Global segment length calculation to create sliding window */

**Algorithm OPS** (trg_set,no_of_cols)

C = (c1, c2, c3,...cn) //set of relevant contexts
W_size=0
while (!NULL(trg_set))
   For each context, c in C
      durationsc = Duration of all occurrences of c
   end while
For each context, c in C
   min_durationsc = MIN (durationsc)
/*take average of minimum lengths of each context */
w_size = MEAN (min_durations);
print("Window Size is ",w_size);
/* calculate overall minimum actual length of any context*/
all_min = MIN (min_durations);
/* do an overlapping of difference of average and minimum actual length*/
/* Initialize, ss - sensor stream at time instant t, a row of size no_of_cols
olap_size = w_size - all_min

/*/ initialize first segment */
seg0 = {ss1,...,ssw_size}
end_pt0 = w_size
prev_seg= seg0
/* starting and ending point of a subsequent segment 'i' */
st_pti= end_pti-1-olap_size;
end_pti = st_pti + w_size-1
/* divide further data as overlapped segments */

While (sensor_readings)
    segi = { ssst_pti, ssst_pti+1, ........ ssend_pti } 
end while
Computation complexity of OPS algorithm for window size detection is $O(n)$ and can be done in real time where $n$ is the number of tuples in ground truth dataset. Proposed algorithm was compared with sliding window of various sizes and varying percentages of overlapping. Results of comparison are shown in Figure 4.6 to 4.8.

![Figure 4.6: Mean ACF Using Different Window Sizes of Segments of Lower Body Sensor Data in Opportunity Dataset](image)

Auto correlation among segments of commonly used sizes was compared with segments having size calculated using OPS algorithm. Dataset of lower body sensors taken from D1 producing streams of 42 raw values in each instance with ground truth annotations of ‘locomotion’ of person being monitored was used for optimal duration calculation of each segment. The optimal duration was calculated as 5 seconds.

![Figure 4.7: Convergence of ACF with Optimal Window Size](image)
Figure 4.6 shows autocorrelation plots of segments of popularly chosen sizes and the optimal calculated size. It can be observed that the optimal size gives the highest correlation among the segments of lower sizes.

Optimal segmentation was further compared with segment autocorrelation obtained with higher window sizes. Results obtained are shown in Figure 4.7. Unlike the trend in Figure 4.6, it can be observed that there is no significant increase in autocorrelation values beyond the optimal size. Thus there is beyond optimal segment size, there is an increase in cost of segmentation (delay) without any improvement in quality of each segment. Thus it is concluded that segments obtained by OPS algorithm are most coherent and appropriate for segmentation.

![Figure 4.7: Mean ACF on Different Inter Segment Overlapping](image-url)

Another parameter to be decided for accurate segmentation is the amount of overlapping samples between consecutive windows. In Figure 4.8, effect of overlapping of samples on segment quality was investigated. Segment size of optimal duration was used in this study. It can be seen that different percentages of overlapping do not affect segment quality in terms of ACF.

Overlapping is an overhead in calculation. With optimal sized windows, one can do away with overlapping without compromising segment quality. However, the effect of overlapping in
accuracy of target context classification is still to be investigated in later chapters. After window size calculation, next processing steps are done on segmented data.

4.4. OUTLIER DETECTION AND FEATURE EXTRACTION

4.4.1. Outlier Detection in a Segment
The segmented data is analyzed for outliers before extracting features from it. A data instance in a segment is said to be an outlier if it significantly deviates from other data points in same segment. The source of outliers could be erroneous measurements or anomalous conditions.

Influence of outliers on further processing can be minimized by two steps namely detection and replacement. Outlier detection is important to control the quality of measured data so as to reduce the overhead of transmitting erroneous data. Conventional techniques of outlier detection applied in data mining are not suitable for WSNs due to their resource constraints and distributed stream nature of data. Thus, outlier detection for WSNs should use minimal amount of energy and memory to have reasonable computation and storage requirement. The technique should not require any prior knowledge on data distribution.

One such method that satisfies both requirements is “3δ edit rule”. According to this method any observation that deviates more than three times the standard deviation from the mean is termed as outlier. This method has limitations of not scaling well as soon as percentage of outliers becomes more than 10% [175]. Some methods combine information from current readings of similar spatially co-located nodes along with its own recent past values. These approaches have been found to achieve high accuracy of outlier detection due to harnessing correlations in multiple variables [124]. However, though analysis of multivariate data improves the accuracy of outlier detection techniques, it also increases computational complexity. Therefore, for on-node processing, simple uni-variate outlier detection techniques have been utilized. Multi-variate outlier detection will be applied at the sink, which has more resources than sensors.

For this study, segment specific outliers are identified and replaced. The outlier treatment is a two-step process, detection of outliers and their replacement. A commonly used method, Hampel
identifier [176] has been used for detection of outliers. The nominal data range for this outlier detection procedure is:

\[
    f(\text{value}) = [\text{median} - c \times \text{MAD}, \text{median} + c \times \text{MAD}]
\]  

(4.6)

In eq. 4.6, MAD is Median Absolute Deviation of segment and ‘c’ is a constant generally chosen to be 3 for normally distributed data. If value falls outside range of \( f(\text{value}) \) then it is considered to be an outlier. This outlier detection method is robust as it will identify transient outliers and will not affect bursty ones which may be an actual abnormal condition. Outlier detection by Hampel identifier was compared with other simple techniques of average and standard deviation based outlier detection.

**4.4.2. Feature Extraction from a Segment**

Feature extraction is about representing each sensor wise segment of data by single value such that the derived value is more informative and non-redundant for subsequent fusion steps. No classifier of whatever sophistication can give reasonable results with features that have no discriminant power. Extracting features thus also reduces the cost of transmission leading to energy savings.

Sometimes feature extraction has been related to dimensionality reduction also [177]. In feature extraction for dimensionality reduction, values from more than one dimension of data are combined to obtain reduced total number of dimensions. The reduced set of dimensions is termed as “set of features”.

However, in order to have a sensor independent feature representation and retain all dimensions of original data, one feature per dimension has been extracted in this work. That is, feature extraction compresses the data only horizontally and not in number of dimensions. The features were extracted for each segment of each sensor data dimension.

Simple mathematical and statistical metrics can be used to extract basic signal information from raw sensor data. Several time domain features which have low computational complexity were
investigated. The choice of features has been guided by resource and power constraints of sensor and real time requirements of processed data.

For each segment \( seg(D_{t_1:t_2}^i) \) of every ‘i’ dimension of data, following statistical features have been investigated in this study:-

a) **Mean** over a segment is a meaningful characteristic for all sensors irrespective of their type. Moreover, with \( O(W) \) complexity, it is simple to compute online and has modest storage requirement. Mean of a segment is calculated as per eq. 4.7.

\[
\text{Mean} \ (seg(D_{t_1:t_2}^i)) = \frac{1}{W} \sum_{j=t_1+1}^{t_2} d_j^i
\]  

(4.7)

It is not possible to apply mean for categorical and binary valued real or virtual sensors. Featured values can be approximated from segments of such sensor by finding mode. Mode of each segment is the most frequently occurring value within that segment. Like mean, mode also has linear computational complexity.

b) **Standard Deviation** (Stdev) - Stdev is defined as the square root of average of the squared differences of each data element from segment mean. This metric can help identify different types of target actions as its values indicate the stability of sensor data. Standard deviation (stdev) of each segment, \( seg(D_{t_1:t_2}^i) \), can be calculated as per eq. 4.8.

\[
\text{Stdev}(seg(D_{t_1:t_2}^i)) = \frac{1}{W} \sqrt{\sum_{j=t_1+1}^{t_2} (d_j^i - \text{Mean} \ (seg(D_{t_1:t_2}^i)))^2}
\]  

(4.8)

Abstract computational complexity of calculating standard deviation is also \( O(W) \). The exact number of addition operations to be done are however, \((2*W + 1)\). \((W+1)\) multiplications will be required to be done by underlying hardware.

c) **Median Absolute Deviation** (MAD) is also a good measure of segment variability useful to segregate each target class from each other. It is calculated as the median of deviation or difference between actual value of each instance of a segment and segment median. MAD of a given segment, \( seg(D_{t_1:t_2}^i) \) is expressed by formula in eq. 4.9.

\[
\text{MAD} \ (seg(D_{t_1:t_2}^i)) = \text{median}_{1 \leq j \leq W} \left( |d_j^i - \text{median} \ (seg(D_{t_1:t_2}^i))| \right)
\]  

(4.9)
Computational cost of calculating MAD over each segment is $O(W^2)$. Actual number of logical operations to be done is $W^2(W+1)/2$ comparisons.

d) Range statistically summarizes a univariate data segment as the difference between maximum and minimum values within the segment. Range can help distinguish concepts with similar patterns with difference only in intensities, for example walking and running. It is calculated as per eq. 4.10.

$$Range\ (seg(D^l_{r_1:r_2})) = \max_{0 \leq j \leq W}(d^l_j) - \min_{0 \leq j \leq W}(d^l_j)$$

(4.10)

Range has been tested as one of the features due to its low computational cost of $O(W)$. The actual number of operations to be done in each segment is n comparisons each for finding maximum and minimum making total $2nW$ comparisons.

e) Inter-Quartile Range (IQR) is a measure of dispersion of data, which is more useful than standard deviation when data is not normally distributed. IQR is also considered to be more robust than mean and standard deviation when outliers persist in underlying data. It is calculated as per eq. 4.11.

$$IQR\ (seg(D^l_{r_1:r_2})) = Q_3 - Q_1$$

where, $Q_1$ is the first quartile and is calculated as the number lying in middle of minimum and median of a segment

$Q_3$ is the third quartile and is calculated as the number lying in middle of median and maximum of a segment.

IQR is computationally expensive as minimum, maximum and median of a segment are required to be calculated for calculating it. Overall complexity of computation is $O(W^2)$.

The input to this step was segmented sensor data and output is the set of sensor wise extracted features with data rate of $\text{len}(D^l)/W$. Features described above have been selected for study on the basis of their low computation times for real time implementation. Main purpose of feature extraction from sensor data is to facilitate higher level fusion. Hence, accuracy of fusion is an
important measure to establish correctness of feature used. Chosen features will be evaluated on this measure in next chapter. For now, the features have been compared against the quality of segment approximation obtained by each. Quality can be judged by relative approximation error defined by eq. 4.12. Let \( EF(\text{seg}(D_{t_1:t_2}^{i})) \) be the output of feature extraction using any of the above defined feature. The Relative Approximation Error (RAE) is defined as eq. 4.12.

\[
RAE(EF(\text{seg}(D_{t_1:t_2}^{i}))) = 1 - \frac{EF(\text{seg}(D_{t_1:t_2}^{i})))}{\text{seg}(D_{t_1:t_2}^{i})} \tag{4.12}
\]

The relative approximation error is the ratio of absolute approximation error and actual value of a sample. To obtain the error in percentage for comparison across different sensor value ranges, Normalized Relative Approximation Error (NRAE) as per eq. 4.13 has been used for feature comparison.

\[
NRAE(EF(\text{seg}(D_{t_1:t_2}^{i}))) = \frac{RAE(EF(\text{seg}(D_{t_1:t_2}^{i}))) - \min((EF(\text{seg}(D_{t_1:t_2}^{i})))}{\max((EF(\text{seg}(D_{t_1:t_2}^{i}))) - \min((EF(\text{seg}(D_{t_1:t_2}^{i}))))} \tag{4.13}
\]

Features described in this section were applied on datasets of all subjects of D1. Lower body sensors with total 42 attributes were used in this study. Mean NRAE obtained for each subject and each feature has been shown in Figure 4.9. Optimal window size of 5 seconds obtained in last experiment was utilized for segmentation.

![Figure 4.9: Normalized Relative Approximation Error of Feature Extraction of Different Subjects](image)
It is observed that overall best approximation is obtained for Subject S2. This is due to the fact that there are very few sudden variations in dataset of S2. Thus, the approximation done by almost all features is reasonable and has low approximation error. Datasets of Subjects S3 and S4 have comparatively larger error due to inter-segment variability actions.

![Figure 4.10: Feature Wise Normalized Relative Approximation Error](image)

Figure 4.10 depicts the mean approximation error of each feature across the subjects. This figure facilitates the direct comparison of features on NRAE metric. It can be seen that mean and range almost give same overall approximation error, while MAD gives the worst. IQR also gives comparable performance as intra-segment outliers have already been removed. Conclusively, mean being one of computationally most simple feature, it is preferred over range as generic feature for further fusion.

### 4.5 DISCRETIZATION OF FEATURES

Discretization is last significant step in local processing of sensor data. As described in Chapter 2, higher level of sensor fusion often makes use of machine learning based classification schemes. Most of these techniques work on any of ordinal, nominal or continuous sensor features. A few of them can also work on input data of mixed types of any of these. However, it has been shown that better classification accuracy is obtained with these techniques, if all of the input data is discrete or nominal. For rule based classifiers, rules formed with discrete values are
shorter and simpler to understand [178]. There are many classification algorithms that can handle only discretized data.

Discretization is the process of transforming continuous valued extracted features to set of nominal values. Discretization can be uni-variate or multi-variate. Univariate discretization discretizes one continuous feature at a time while multivariate discretization takes into account several features simultaneously. Univariate discretization methods have been mainly considered here, as data from all sensors are considered independently. In both cases, discretization process is about finding suitable cut points in the data. A cut point or split point is a value in a continuous valued sorted that divides the list in two sub-lists. For example, range of values \([a:b]\) can be partitioned into sub-list \([a:c]\) and \([c:d]\). Value \(c\) is then called a cut point while \([a:c]\) and \([c:d]\) are called its intervals. A typical discretization process recursively finds cut points in data until a stopping criterion is achieved. Different algorithms differ in stopping criteria.

At the end of discretization process, for any sensor data vector, \(D^i\), set of \(x\) cut points, \(\theta = [\theta_0^i, \theta_1^i, \ldots, \theta_m^i]\) are obtained. That is, the entire vector can be represented by \(x\) unique values as represented in eq. 4.14.

\[
[D^i] = \begin{cases} 
0 & \min(D^i) < D^i < \theta_0^i \\
1 & \theta_0^i < D^i < \theta_1^i \\
\vdots & \theta_x^i < D^i < \max(D^i) \\
x & 
\end{cases}
\]  

(4.14)

The set of cut points \(\theta\) and arity \(x\) are different for each sensor and is applied accordingly. Thus, the problem of discretization is to find optimal \(x\) and corresponding \(\theta\).

Manual cut - points can be decided in cases where sensors represent physical parameters that are well understood by human beings and their quantization has universal acceptance. For example, dataset D3 (Refer Appendix A) is real valued depicting the measurements in Celsius for temperature, percentage for relative humidity and Lux for light. These values can be binned to discrete categories by taking inputs from various weather experts and information on Internet.
The categorized symbolic values as shown in Table 4.1 are obtained. Temperature was divided into 5 classes, humidity in 4 classes and light in 7 cut-points. The detail of range of values within each cut-point is also given in Table 4.1.

Table 4.1: Experts’ Quantization of Sensor Data

<table>
<thead>
<tr>
<th>Feature of Context</th>
<th>Class No.</th>
<th>Range</th>
<th>Symbolic Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Temperature (Range in Degree Celsius)</strong></td>
<td>1</td>
<td>&lt;10</td>
<td>Very Cold</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10 – 18</td>
<td>Cold</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>19-25</td>
<td>Normal</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>26-35</td>
<td>Mild</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>&gt;35</td>
<td>Hot</td>
</tr>
<tr>
<td><strong>Humidity (Range in % of Relative Humidity)</strong></td>
<td>1</td>
<td>&lt;=20</td>
<td>Dry</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21-28</td>
<td>Comfortably Humid</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>29-45</td>
<td>Quite Humid</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>&gt;45</td>
<td>Highly Humid</td>
</tr>
<tr>
<td><strong>Ambient Light (in Lux)</strong></td>
<td>1</td>
<td>&lt;=10</td>
<td>Pitch Dark</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11-50</td>
<td>Very Dark</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>51-200</td>
<td>Dark Indoors</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>201-400</td>
<td>Dim Indoors</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>401-1000</td>
<td>Normal Indoors</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>&gt;1000</td>
<td>Bright Indoors</td>
</tr>
</tbody>
</table>

Manual discretization is however, domain specific and cannot be scaled or generalized. For certain sensors like accelerometers no straightforward universal quantization is possible. Thus algorithmic discretization has to be done by learning parameters from available data.

Several discretization algorithms have been proposed in literature [178] [183]. These can be broadly divided as supervised and unsupervised ones:

- The methods that do discretization independent of the target classification information are called unsupervised discretization methods. Classical methods of equal frequency and equal width binning are popular ones in this category.
On the other hand, some methods take into account the class labels for discretizing featured data. These are called supervised discretization methods and popular examples are Boolean reasoning and methods based on entropy minimization.

The supervised methods have proved to be more accurate while unsupervised ones are faster. As both types have their own advantages, representative algorithms from both categories were studied. Some well-known unsupervised discretization techniques analyzed here are as given below. For ease of representation in evaluation, the algorithms are numbered from U1 to U3.

- **U1**: *Uniform Binning* or *Equal Interval Width Binning* - This method works by sorting the vector data and dividing the data values into equally spaced bin ranges. The number of total number of bins, k, is input by user. This method is one of the simplest, but is sensitive to outliers. For example, given the extracted feature values
  \[ f = 0, 1, 2, 3, 5, 10, 100 \]
  and using \( k = 4 \) gives a bin width of \( \frac{100 - 0}{4} = 25 \), resulting in discretization intervals \([0 − 25], [25 − 50], [50 − 75], [75 − 100]\). So all values except 100 fall into the first bin, which is not reasonable.

- **U2**: *Histogram Binning* or *Equal Frequency Interval Binning* overcomes the limitation of U1. In this method also the number of total bins is input by user. After sorting, the whole data is partitioned such that same number of instances is allocated to each bin.

Algorithms U1 and U2 are simple to implement but have some problems. Firstly, the number of intervals has to be specified beforehand. Generally exact value of ‘\( m \)’ is not known beforehand, thus either a random value is fixed or trial and error is used to find an optimal value.

- **U3**: *Proportional K- Interval Discretization (PKID)* [179] – This method tries to find optimal intervals without class information. It considers adjusting cut-points and arity as a function of size of data vector. That is, any given sensor data vector \( D^i \), is discretized into \( \sqrt{\text{len}(D^i)} \) intervals where each interval has \( \sqrt{\text{len}(D^i)} \) instances. As length of \( D^i \) increases, the number of cut points also increase.

This method is simple and improved as compared to earlier two unsupervised methods.

Unsupervised methods do not make use of the class in setting discretization interval values and as a result instances belonging to different classes may be put in same interval. Thus recently,
few supervised discretization methods have also been proposed in literature [178]. Out of all, two prominent ones based on their popularity have been analyzed. For further reference, the supervised algorithms are numbered as S1 and S2.

- **S1**: Recursive Minimal Entropy Partitioning (RMEP) [180] aims to find intervals that minimize the class information entropy. The method uses class wise entropy of each possible partition to select a cut point. New partitions are kept on creating till the time entropy can be further reduced. For sensor dataset $D^i$ and a cut point $c$, the class information entropy induced by $c$ is given as eq. 4.15.

$$\text{Ind}_{Entropy}(D^i,c) = \frac{|D^i_{1:c}|}{|D^i|} \text{Entropy}(D^i_{1:c}) + \frac{|D^i_{1:|\text{len}(D^i)|}|}{|D^i|} \text{Entropy}(D^i_{1:|\text{len}(D^i)|})$$  

(4.15)

where, $D^i_{1:c}$ and $D^i_{1:|\text{len}(D^i)|}$ are sub-lists formed by cut-point $c$

Entropy($D^i_{1:c}$) is the class entropy of a set of values $D^i_{1:c}$ calculated as in eq. 4.16.

$$\text{Entropy}(D^i_{1:c}) = -\sum_{j=1}^{L} p(\alpha_j, D^i_{1:c}) \log_2(p(\alpha_j, D^i_{1:c}))$$  

(4.16)

where, $(\alpha_1, \alpha_2, ..., \alpha_L)$ is the set of possible class labels for any data instance in $D^i$ and $p(\alpha_j, D^i_{1:c})$ is the probability of occurrence of a class label $\alpha_j$ in $D^i_{1:c}$.

A cut point, $c$, which minimises Entropy($D^i_{1:c}$) is considered optimal. The recursive partitioning is stopped when a Minimum Description Length (MDL) is obtained. MDL is encoding length of the obtained intervals.

- **S2**: Kononeko’s MDL method is similar to S2 except that it provides a better encoding by correcting entropy measure bias in a multi-valued attribute [181].

It can be deduced from the description of working of various algorithms that learning discretization parameters requires all the archival data with or without class information. Thus discretization cut-points are learnt at sink during training. The learnt boundaries are then
communicated to each sensor, which are used there for online discretization. The effect of already mentioned five methods on context extraction accuracy at the context heads will be studied in next chapter. The statement is supported by results presented in last section.

### 4.6 Analytical Evaluation of Energy Efficiency

In sensor networks, energy consumption for data propagation on networks is manifold higher than the energy consumption for on board processing [182]. Therefore, reducing transmission leads to significant energy efficiency. Power ratio, which is defined as the fraction of energy consumption between sending processed data and sending raw data is a metric for measuring reduced power consumption.

Processing steps like segmentation and feature extraction reduce the number of packets transmitted to sink. Fewer transmitted packets incur a low communication overhead, leading to energy-efficiency.

The energy consumed by each node while transmitting raw data is represented as \( E_{\text{raw}} \) and that by processed data as \( E_{\text{reduced}} \). In sequential programming model, which is the case here, the energy consumption of an algorithm is directly proportional to time taken for computation.

\[
E_{\text{raw}} = (E_{\text{sens}} * n * m) + (E_{\text{mem\_access}} * n * m) + E_T * n * m \text{ J/sec} \tag{4.17}
\]

where \( E_T \) = Energy consumption in per sample data transmission, \( n \) and \( m \) are the number of samples per sec and attributes per sample respectively.

\( E_{\text{sens}} \) and \( E_{\text{mem\_access}} \) are the energy consumed in sensing and read-write memory access respectively per sample data.

\[
E_{\text{reduced}} = (E_{\text{sens}} * n * m) + E_{\text{feature}} + E_{\text{disc}} + (n/w) * m * E_T \tag{4.18}
\]

where \( w \) is the window size for feature extraction and a constant fraction of sample size \( n \).

\[
E_{\text{feature}} = 2*w * E_{\text{mem\_access}} + 2*w * E_{\text{proc\_add}} + E_{\text{proc\_div}} \tag{4.19}
\]

\( E_{\text{proc\_add}} \) and \( E_{\text{proc\_div}} \) are processing costs and can be considered \( 1/8^{\text{th}} \) of the transmission cost in sensors, Thus (4.19) becomes

\[
E_{\text{feature}} = 2*w * E_{\text{mem\_access}} + (2*w +1)* 0.125* E_T \tag{4.20}
\]
\[ E_{\text{disc}} = \log_2 D \* E_{\text{mem_access}} = \log_2 (n/c_1) \* E_{\text{mem_access}} \]  

(4.21)

Using (4.20) and (4.21), (4.19) becomes

\[ E_{\text{reduced}} = (E_{\text{sens}} \* n*m) + 2*n/c_2 \* E_{\text{mem_access}} + (2*w +1)* 0.125*E_T \]

\[ + \log_2(n/c_1) \* E_{\text{mem_access}} + (n/ w) *m* E_T \]  

(4.22)

On simplifying,

\[ E_{\text{reduced}} = (E_{\text{sens}} \* n*m) + (2*n/c_2 + \log_2(n/c_1)) \* E_{\text{mem_access}} + (2*w +1)* 0.125*E_T + (n/ w) *m* E_T \]  

(4.23)

Energy gain due to difference in energy consumed in sending raw data from that in sending processed data per second is then calculated as eq. 4.24.

\[ E_{\text{sens_gain}} = E_{\text{trg}} - E_{\text{reduced}} = ((E_{\text{sens}} \* n*m) + (E_{\text{mem_access}} \* n*m)*E_T) \cdot (E_{\text{sens}} \* n*m) + (2*n/c_2 + \log_2(n/c_1)) \* E_{\text{mem_access}} + (2*n/c_1 +1)* 0.125*E_T + (n/ c_1) *m* E_T \]  

(4.24)

where \( E_{\text{sens_gain}} \) is the Energy Gain per Sensor Node

\[ E_{\text{global_gain}} = \sum_{i=1}^{N} E_{\text{sens_gain}} \]  

(4.25)

\( E_{\text{global_gain}} \) is the total energy gain, if \( N \) sensors are present in the system.

Thus, substantial energy savings can be obtained by local processing at sensor nodes.

**4.7 CHAPTER SUMMARY**

Sensor data processing steps on-node have been defined in this chapter to eliminate redundancies of raw data and at the same time preserving characteristics of original information. Exhaustive testing on different alternatives for all processing sub methods has been done to find most optimal combination of all parameters to be used in actual implementation. Various combinations of parameters to be used in evaluation are given in Table 4.2. The experiments were conducted to exhaustively assess the most optimal values for all processes.

For all experiments, data from lower body sensors of Opportunity Dataset D1 has been utilized.

As a result of study done in this chapter, following parameters of different algorithms are learnt:

a) Window size and overlapping for segmentation
b) Outlier detection method

c) Feature to be extracted for each sensor

d) Cut- points for Discretization

**Table 4.2: Methods Evaluated for Various Processing Steps**

<table>
<thead>
<tr>
<th>Segmentation (SE) SW size (in seconds), Overlapping</th>
<th>Outlier Detection</th>
<th>Outlier / Missing Value Replacement</th>
<th>Extracted Features</th>
<th>Discretization Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE1: 1, 25%, 50%, 75%</td>
<td>OD1: +Average of values in window</td>
<td>RE1: Median</td>
<td>1) Mean</td>
<td>U1: Uniform Binning</td>
</tr>
<tr>
<td>SE2: 2, 25%, 50%, 75%</td>
<td>OD2: ±3 *Stddev</td>
<td>RE2: Mean</td>
<td>2) Range</td>
<td>U2: Histogram Binning</td>
</tr>
<tr>
<td>SE3: 3, 25%, 50%, 75%</td>
<td>OD3:[Median-c* MAD, Median+c* MAD]</td>
<td>RE3: Mode</td>
<td>3) Inter Quartile Range (IQR)</td>
<td>U3: Proportional K-Interval Discretization (PKID)</td>
</tr>
<tr>
<td>SE4: 5, 25%, 50%, 75%</td>
<td></td>
<td>RE_OUR: Repetition of last available value</td>
<td>4) Mean Absolute Deviation (MAD)</td>
<td></td>
</tr>
<tr>
<td>SE 5: Proposed Heuristic: OPS</td>
<td></td>
<td></td>
<td>5) Standard Deviation (Stdev)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6) Mode for categorical variables</td>
<td></td>
</tr>
</tbody>
</table>

These parameters are used by sensor nodes to process the data before communication to next hop device. The storage requirement of parameters is very modest and can be easily stored in the sensor node itself. In next chapter quantitative comparisons based on predictive accuracy using next level fusion classifier will be done.