Event Classification and Filtering of False Alarms in Wireless Sensor Networks*

Markus Wälchli and Torsten Braun
Institute of Computer Science and Applied Mathematics
University of Bern - Switzerland
{waelchli,braun}@iam.unibe.ch

Abstract

In this paper the classification of discrete events, computed on tiny wireless sensor nodes, is investigated. Three different classifiers are evaluated: a Bayesian classifier, a fuzzy logic controller (FLC), and a neural network approach. The target applications pose several requirements on the classifiers. No a priori knowledge about the event classes is available. Events are only observable as collections of raw sensor data. Accordingly, event classes need to be learned from that raw (training) data. As a consequence, pre-labeling of the events is not possible either. In our work, event classes are learned by a k-means clustering algorithm. Any subsequent classifier training is based on these extracted event classes. Thus, the resulting classifiers are completely self-learning. Event classes are learned from emitted signal strength estimations, which are collected and processed by dynamically established tracking groups. The resulting event estimates are reported to a base station, where the classifiers are trained. The learned classifier parameters are then downloaded onto the sensor nodes, where any subsequent classification and filtering is performed.

1 Introduction

Sensor networks consist of tiny, battery-powered sensing devices, equipped with some processing power, some storage, a radio, and an array of sensors to monitor the physical environment. Environmental data is sensed and processed on the sensor nodes. In general, observation (event) reports are sent to a base station, where the data is stored and/or might be further processed.

Reporting all measurements from all sensor nodes to the base station is expensive. Accordingly, distributed algorithms for data processing and aggregation are needed. Moreover, if the system is able to filter false or non-relevant event reports, communication costs can be saved and false alarms prevented. To evaluate and filter event reports, means to distinguish different event types, as well as confidences in the classifications, are required.

In this work, classifiers modeled based on measured data are considered. This means the event classes and the parameters of the classifier are learned, respectively estimated, from training data. In many applications, i.e., most monitoring systems, it would be difficult to predetermine the event classes. Accordingly, unsupervised learning techniques are needed. Learning from data has the advantage that no expert knowledge is required. Thus, the application of the algorithm is simpler and design flaws due to poor data abstractions can be prevented.

Another classifier requirement is the ability to filter suspicious data (e.g., false alarms). A classifier based on fuzzy logic concepts and a neural network approach are proposed. The classifiers assign a confidence degree to each classification. Thus, it is possible to filter events which do not satisfy a given threshold requirement. As non-satisfying event reports are filtered, a periodic reclassification of events is needed. This is supported in many applications anyway, in particular if events need to be tracked. Obviously, filtering event reports has an impact on reporting delays. On the other hand, preventing false alarms saves costs in terms of energy and money. The trade-off between false-alarm tolerance and reporting latency is investigated.

The detection and classification of events is done by collaborative signal processing (CSP) in a spatially restricted area. An event is detected and tracked by dynamically established groups, which are lead by a dedicated node. Measurements of the group members are gathered at this leader node. In the current implementation the leader estimates the emission powers of multiple signals emitted by an event source. The classifier is based on these estimates. All communication and computation is performed locally and only the satisfying event reports are routed to the base station.

In contrast to classification, which can easily be executed on sensor nodes, the estimation of the classifier parameters,

*The work presented in this paper was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

i.e., the tuning of the classifier, is performed at a base station. This has two reasons: First, the classifier is tuned from training data which needs to be collected at a central instance. Second, the tuning of the classifier is too expensive to be run on a simple sensor node, but can be performed at a base station equipped with more computation power. Alternatively, if the event reports are routed to a dedicated station in the Internet, the classifier tuning can be performed there too. Once the parameters of the classifier have been computed, they only need to be downloaded onto the sensor nodes, which perform any subsequent classification.

In Section 2 related work is discussed. Section 3 introduces the clustering mechanism and the different classifiers. The generation of training and test sets is addressed in Section 4. The evaluation of the classifiers is provided in Section 5. Section 6 concludes the paper and gives an outlook.

2 Related work

Related work can mainly be divided into three fields. There are approaches in event detection and tracking group organization, providing primitives to collect data and to organize event-observing areas. Second, event classification algorithms, which commonly focus on improving the classification accuracy. In contrast, our focus is on classification error control and false alarm prevention. Third, there is theory in fuzzy classifier and neural network design. This has been well studied in other research fields, but has not yet gained much attention in wireless sensor networks.

Detection and tracking group formation issues have been widely investigated. [6] organizes groups based on consensus. This requires mutual negotiation implying high communication costs. In [8] mobile targets (events) are considered. The group organization is based on estimated target velocities. In-group communication is done in a message-passing-like manner, implying again rather high communication costs. EnviroSuite [9] is a distributed event tracking algorithm. It supports event detection and tracking, but neither localization nor classification. The tracking groups are dynamically established. The group is maintained by a simple periodic heartbeat message scheme. Our networking approach (DELTA) [16] provides similar basic operations, but adds additional features such as event classification. Another event detection scheme has been proposed in [1]. IDSQ incrementally queries group members until a belief state is satisfied. The incremental querying implies delays, though. In other research querying systems have been enhanced to support distributed event detection. For example, [13] proposes a declarative query language based on discrete events. Querying systems basically require a priori system knowledge and experts.

Event classification with wireless sensor nodes has been addressed in the SensIt project (e.g., [8], [15]). Statistical methods based on time series of event measurements have been investigated. All approaches share a rather centralized nature requiring high communication costs. Non of the approaches provides means for data filtering and classification error-rate control. In [12] events are classified by applying a decentralized incremental subgradient optimization method, where a parameter estimate is incrementally updated until a precision mean is satisfied. Again, no data filtering is supported. In [17] a fault-tolerant classifier based on local binary decisions is introduced. The approach depends on error-correcting codes. Binary decisions are collected at fusion centers, where the classification is performed. A priori knowledge of the events and precomputation of the codes are required. In [3] vehicles, persons, and persons equipped with ferrous objects are classified based on a decision tree. All decisions are based on thresholds, requiring expert knowledge.

Dimensionality reduction techniques such as multidimensional scaling, nearest neighbor search, and principal component analysis (e.g., [4], [10]), are related to classification problems. In [5] the application of ART neural networks for dimensionality reduction in wireless sensor networks has been proposed. ART neural networks learn new patterns online. Although the parametrization of the system might be difficult, an integration of ART neural networks could be promising. A good introduction to fuzzy classifier design is the work of Kuncheva [7]. Some of the design decisions considering the proposed FLC classifier have been inspired by this book. The challenge is to compose the appropriate model for the present kinds of events, though.

3 Event Classification

Unsupervised classification algorithms are considered. These algorithms can be tuned directly from data, requiring only limited expert knowledge and no a priori knowledge of the system. Obviously, training data must be available. This data might be generated in specific training phases or could, alternatively, be extracted from the event reports continuously collected at the base station. In the latter case, event reports should not be filtered in the network, but at the base station. Else, the classifier tuning would be applied on filtered data. A retraining of the classifier might be necessary if, for example, new event patterns occur. The retraining can be performed on demand or periodically.

3.1 Extracting the Event Classes

In unsupervised learning, event classes are modeled based on training data. In this work two well-known clustering mechanisms, k-means and fuzzy k-means, have been used. Clustering algorithms arrange samples into groups

(clusters) according to some similarity metrics, e.g., the Euclidean distance, or membership degrees.

The goal of clustering is to find a decomposition of the (training) set $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_M\}$ into m clusters $\{C_1, ..., C_m\}$. The basic idea of fuzzy k-means is to assign each $\mathbf{z} \in \mathbf{Z}$ to each cluster C_j with a given membership degree $\mu_j(\mathbf{z})$. Hard clustering (k-means) is a special case of fuzzy clustering with $\mu_j(\mathbf{z}) \in \{0, 1\}$ for all j. The fuzzy k-means algorithm requires the computation of the membership degree of a sample $\mathbf{z}$ according to [2]:

$$\mu_j(\mathbf{z}) = \begin{cases} 1, & \text{if } \mathbf{z} = \mathbf{m}_j, \\ \frac{1}{\sum_{k=1}^{K} \left(\frac{\|\mathbf{z}-\mathbf{m}_j\|}{\|\mathbf{z}-\mathbf{m}_k\|}\right)^{\frac{2}{\beta-1}}}, & \text{else.} \end{cases} \quad (1)$$

where β is a parameter controlling the gradient of the membership and $\|.\|$ is the Euclidean norm. According to (1) the membership degree of a sample $\mathbf{z}$ is the higher, the closer $\mathbf{z}$ and a cluster center $\mathbf{m}_j$ are. The computation of the cluster centers $\mathbf{m}_j$ itself is based on the membership degrees of all samples $\mathbf{z} \in \mathbf{Z}$ [2]:

$$\mathbf{m}_j = \frac{\sum_{i=1}^{M} \mu_j(\mathbf{z}_i) \cdot \mathbf{z}_i}{\sum_{i=1}^{M} \mu_j(\mathbf{z}_i)} \quad (2)$$

Having defined these preliminaries, the fuzzy k-means algorithm can be written as:

Fuzzy k-means

input: Training set $\mathbf{Z}$; K = number of clusters;
ouput: m clusters $\{C_1, ..., C_m\}$;
$\mu_j(\mathbf{z}_i)$ for $1 \leq j \leq K$ and $1 \leq i \leq M$;
begin
choose K initial cluster centers $m_1, ..., m_K$;
repeat
compute $\mu_j(\mathbf{z}_i)$ for $1 \leq j \leq K$ and $1 \leq i \leq M$ according to (1);
update the cluster centers $\mathbf{m}_j$ for $1 \leq j \leq K$ according to (2);
until termination criteria satisfied;
end

The fuzzy k-means algorithm works similar to the k-means algorithm. The main difference is that it does not fixly assign a sample to a cluster, but assigns a sample to a cluster with a certain membership degree. This has also some impact on the computation of the cluster centers, as the impact of all samples is regarded instead of only considering the samples belonging to the specific cluster.

Having applied any of the two clustering algorithms on a training set, the classifiers discussed in the next section can be tuned based on the learned clusters (classes). It remains to be highlighted that both clustering algorithms can be used as substitutes and produce similar clusters. K-means has been used to tune the Bayesian classifier and the neural network, while the FLC was tuned with fuzzy k-means.

3.2 Bayesian Classifier

It is assumed that m different event classes (clusters) C_i have been learned. Each sample $\mathbf{x}$ of C_i is an element of $\Re^n$, where n is the total number of observed phenomena (features). In the current implementation each sample $\mathbf{x}$ consists of two different light values. The feature or phenomena space is accordingly $\Re^2$.

To implement a Bayesian classifier the a priori class probabilities $p(C_i)$, i.e., the probabilities representing the frequencies of classes C_i in the sample, as well as the class-specific probabilities $p(\mathbf{x}|C_i)$ are required. $p(\mathbf{x}|C_i)$ is the probability to which extend $\mathbf{x}$ belongs to C_i. A Bayesian classifier implements the following classification rule [7], [2]:

$$\mathbf{x} \in C_i \Leftrightarrow p(\mathbf{x}|C_i)p(C_i) > p(\mathbf{x}|C_j)p(C_j), \quad \forall j = 1, ..., m; j \neq i \quad (3)$$

In the following normal distributions of the probabilities $p(\mathbf{x}|C_i)$ are assumed. Normal distributions are analytically well manageable. The parameters, i.e., the mean value $\mathbf{m}$ and the standard deviations $\mathbf{K}$ of each cluster, can easily be estimated from the samples in the C_i. The normal distribution in n dimensions, with $n \geq 2$, looks as follows [2]:

$$p(\mathbf{x}|C_i) = \frac{1}{(2\pi)^{\frac{n}{2}} |\mathbf{K}_i|^{\frac{1}{2}}} e^{\left[-\frac{1}{2}(\mathbf{x}-\mathbf{m}_i)' \mathbf{K}_i{}^{-1} (\mathbf{x}-\mathbf{m}_i)\right]} \quad (4)$$

This function only requires the knowledge of the mean vectors $\mathbf{m}_i$ and the covariance matrices $\mathbf{K}_i$ for each class C_i. The covariance matrix is a $n \times n$ matrix. $|\mathbf{K}_i|$ is the determinant of $\mathbf{K}_i$. Instead of using $p(\mathbf{x}|C_i)p(C_i)$ in the classification rule (3), a monotone function can be applied to simplify the computation. As done by other authors [7], [2], a natural logarithm is used:

$$D_i(\mathbf{x}) = ln[p(\mathbf{x}|C_i)p(C_i)] \quad (5)$$

Thus, the new classification rule is:

$$\mathbf{x} \in C_i \Leftrightarrow D_i(\mathbf{x}) > D_j(\mathbf{x}), \quad \forall j = 1, ..., m; j \neq i \quad (6)$$

The Bayesian classifier is fully functional as soon as the the covariance matrix $\mathbf{K}_i$ and the mean $\mathbf{m}_i$ of each cluster C_i are computed. Having samples $\{\mathbf{x}_1, ..., \mathbf{x}_M\}$ of a class C_i, these parameters are estimated as follows [2]:

$$\mathbf{m_i} = \frac{1}{M} \sum_{j=1}^{M} \mathbf{x}_j, \qquad and \qquad \mathbf{K}_i = \sum_{j=1}^{M} \mathbf{x}_j \mathbf{x}_j' - \mathbf{m}\mathbf{m}'$$

The classifier does not support any confidence in its classifications. A sample is fixly assigned to the cluster it belongs to with highest probability.

3.3 Neural Network

Neural networks [2] are bio-inspired networks which, in general, are neither self-documenting nor directly comprehensible for human beings. They consist of simple, mutually connected computing units (neurons), which support parallel computing, learning, and generalization.

In our implementation a multilayer feedforward neural network (FFNN) has been used. Basically, FFNNs consist of an input layer, an output layer, and one or more hidden layers of neurons. The number of input variables, output variables and neurons is arbitrary. A simple neuron is defined as follows:

$$f(x_1, ..., x_n) = \sum_{i=1}^{n} w_i x_i \tag{7}$$

where a weight w_i is assigned to each input variable x_i. In FFNNs these weights are learned from training data consisting of input samples and the associated output. The training is based on the backpropagation method (e.g., [2]), which feedforwards the input training pattern throughout the neural network, analyzes the error by backpropagation, and updates the weights. The backpropagation procedure is based on the popular steepest descent minimization method, which is used to solve nonlinear optimization problems.

The FFNN is trained off-line based on the collected event reports. The output (event class) associated with each sample is determined by k-means clustering. Once trained, only the weights need to be downloaded onto the sensor nodes.

3.4 Fuzzy Logic Controller (FLC)

In this section the FLC is introduced. All computations are based on the clusters learned by fuzzy k-means. We assume again that m different event classes C_i have been extracted. Let $U = \{u_1, ..., u_n\}$ be the universal set. A fuzzy set $\tilde{A}$ on U is described by the membership function:

$$\mu_{\tilde{A}} : U \rightarrow [0, 1] \tag{8}$$

where $\mu_{\tilde{A}}(u)$ expresses the membership degree in which the element u belongs in $\tilde{A}$.

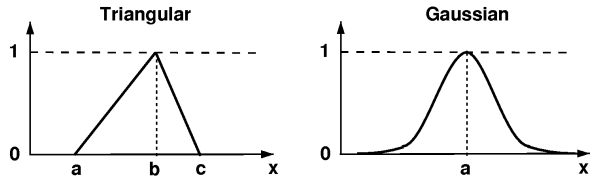

Figure 1. Used membership functions.

In our work, we investigate triangular and Gaussian membership functions. The functions are depicted in Figure 1 and computed according to:

$$\mu(x) = \begin{cases} \frac{x-a}{b-a}, & \text{if x} \in (a, b], \\ \frac{c-x}{c-b}, & \text{if x} \in (b, c], \\ 0, & \text{otherwise.} \end{cases} \quad and \quad \mu(x) = e^{-\frac{(x-a)^2}{2\sigma^2}}$$

Gaussian functions have the advantage that the whole feature space is covered. Considering a classification problem with $\mathbf{x} \in \Re^n$ and m event classes, the proposed fuzzy logic controller consists of a system of m fuzzy if-then rules R_k of the form [7]:

$$R_k : IF \;\; \mu_{\tilde{A}_{k,1}}(x_1) \wedge \mu_{\tilde{A}_{k,2}}(x_2) \wedge ... \wedge \mu_{\tilde{A}_{k,n}}(x_n) \tag{9}$$
$$THEN \;\; g_k(\mathbf{x}) > g_i(\mathbf{x}), \;\; \forall i = 1, ..., m$$

where $\wedge$ is an arbitrary operator aggregating the fuzzy sets of the premises and $g(\mathbf{x})$ is an arbitrary function of the consequence. The rule means that the degree of satisfaction of the premise is assigned to the conclusion. Accordingly, if the premise of R_k is fulfilled to a high degree, then the consequence of R_k has a high degree of significance too. Moreover, considering a sample $\mathbf{x}$, the degree of satisfaction of rule R_k shall be maximum if $\mathbf{x}$ belongs to class C_k. The fuzzy sets $\tilde{A}_{k,i}$ are learned from the clusters and the consequence functions $g(\mathbf{x})$ are modeled according to a Takagi-Sugeno TSK2 classifier [7]. A TSK classifier implements rules of the general form (9) in the following way [7]:

$$R_k : IF \;\; \mu_{\tilde{A}_{k,1}}(x_1) \wedge \mu_{\tilde{A}_{k,2}}(x_2) \wedge ... \wedge \mu_{\tilde{A}_{k,n}}(x_n) \tag{10}$$
$$THEN \;\; g_k(\mathbf{x}) = f_k(\mathbf{x})$$

where $f_k(\mathbf{x})$ is again an arbitrary function. TSK classifiers have a number of properties. In particular their parameters can be estimated from training data and, therefore, no expert knowledge is required. Second, the conclusion assigns a sample with a certain membership degree to a specific class. This allows the assignment of belief degrees to classifications, supporting false-alarm filtering. The specific TSK2 classifier is derived from the generic TSK model by specifying

TSK2 classifier

- $z_{k,i} \in \Re, \;\; k = 1, ..., m, \;\; i = 1, ..., m$;
- The conjunction (AND) is the product;
- The ith output of TSK2 is

$$g_i^{TSK2}(\mathbf{x}) = \frac{\sum_{k=1}^{M} z_{k,i} \prod_{j=1}^{n} \mu_{\tilde{A}_{k,j}}(\mathbf{x}_j)}{\sum_{k=1}^{M} \prod_{j=1}^{n} \mu_{\tilde{A}_{k,j}}(\mathbf{x}_j)} \tag{11}$$

Accordingly, after having determined the fuzzy sets of the premises, the fuzzy consequences $g_i(\mathbf{x})$ can be tuned

for the TSK2 classifier (see [7], p. 182). This means that estimates of the $z_{k,i}$ have to be computed. In the following tuning of premises and the consequences are discussed. The idea is to generate exactly one rule for each cluster. Accordingly, each classification rule is used to classify samples of one specific event class (cluster).

3.4.1 Tuning the premises

The premises are tuned from the clusters learned by fuzzy k-means. Thereby, the fuzzy sets of the premises are the projections of the clusters on the coordinate axes, where the cluster center has the highest membership degree. The projection of a 2-dimensional cluster, i.e., the samples consist of two kinds of phenomena, is shown in Figure 2.

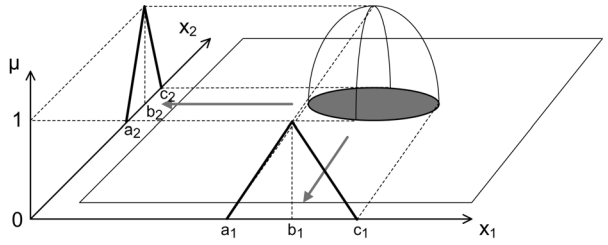

Figure 2. Mapping a cluster to the fuzzy sets $\tilde{\mathbf{A}}_{\mathbf{k,i}}$ of the premises in $\Re^2$.

The rule R_k used to classify samples into this cluster looks as follows:

$$R_k : IF \;\; \mu_{\tilde{A}_{k,1}}(x_1) \wedge \mu_{\tilde{A}_{k,2}}(x_2) \;\; THEN \;\; g_k^{TSK2}(\mathbf{x})$$

Different kinds of membership functions $\mu_{\tilde{A}_{k,i}}$, with $i = 1, .., n$, can be extracted from a cluster. In our work triangular and Gaussian membership functions are used. In order to parameterize a triangular function $\mu_{\tilde{A}_{k,i}}$, the minimum, mean, and maximum values of a cluster in dimension i are needed. The determination of the parameters of a Gaussian membership function $\mu_{\tilde{A}_{k,i}}$ requires the mean and the standard deviation of a cluster in dimension i.

3.4.2 Tuning the consequences

The consequence function g_k^{TSK2} is computed for each rule R_k. Therefore, the degree of satisfaction (significance) of the premise of each R_k, in the following labeled as firing strengths $\tau_k(\mathbf{z})$, needs to be determined:

$$\tau_k(\mathbf{z}) = \Lambda(\mu_{\tilde{A}_{k,1}}(\mathbf{z}_1), ..., \mu_{\tilde{A}_{k,n}}(\mathbf{z}_n)) \qquad (12)$$

For the TSK2 classifier (see Eq. (11)) the product is used as aggregation function Λ.

In order to obtain the firing strength $\beta_{k,i}$, of a rule R_k to a cluster C_i, the sum of firing strengths $\beta_1, ..., \beta_m$ of all samples belonging to that specific cluster C_i are computed according to [7]:

$$\beta_{k,i} = \sum_{\mathbf{z} \in \mathbf{Z}} Ind(\mathbf{z}, C_i)\tau_k(\mathbf{z}), \qquad \forall k = 1, ..., m; \qquad (13)$$

where $Ind(\mathbf{z}, C_i)$ indicates the cluster C_i to which the element $\mathbf{z}$ belongs:

$$Ind(\mathbf{z}, C_i) = \begin{cases} 1, & \text{if } \tau_k(\mathbf{z}) \text{ is maximum,} \\ 0, & \text{else.} \end{cases} \qquad (14)$$

This means having determined the cluster C_i with maximum firing strength as 'class label' of a sample $\mathbf{z}$, the firing strengths of $\mathbf{z}$ in respect to each rule R_k is assigned to the according sum in $\beta_{k,i}$. An example illustrates the procedure. Given the set of samples $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_4\}$ and four rules $R_1, ..., R_4$, one for each cluster $C_1, ..., C_4$, the firing strengths $\tau_1, ..., \tau_4$ of each sample $\mathbf{z}$ to each rule are:

Sample	$\tau_1(\mathbf{z})$	$\tau_2(\mathbf{z})$	$\tau_3(\mathbf{z})$	$\tau_4(\mathbf{z})$	Class
$\mathbf{z}_1$	**0.6**	0.2	0.1	0.1	C_1
$\mathbf{z}_2$	0.1	**0.7**	0.1	0.1	C_2
$\mathbf{z}_3$	**0.4**	0.3	0.1	0.2	C_1
$\mathbf{z}_4$	0.2	0.2	**0.5**	0.1	C_3

Sample $\mathbf{z}_1$, for example, has a firing strength of 0.6 in respect to C_1, of 0.2 in respect to C_2, and of 0.1 in respect to C_3 and C_4, respectively. According to Equation (14), $\mathbf{z}_1$ is therefore assigned to C_1 (bold in Table 3.4.2). Similar assignments are obtained for the other samples. By applying Equation (13) the matrix $\beta_{k,i}$ is filled as follows:

$$\beta_{k,i} = \begin{bmatrix} 0.6+0.4 & 0.1 & 0.2 & 0 \\ 0.2+0.3 & 0.7 & 0.2 & 0 \\ 0.1+0.1 & 0.1 & 0.5 & 0 \\ 0.1+0.2 & 0.1 & 0.1 & 0 \end{bmatrix}$$

In the first column $\beta_{k,1}$ of $\beta_{k,i}$, the firing strengths of $\{\mathbf{z}_1, \mathbf{z}_3\}$, which belong to cluster C_1, are recorded in respect to the according rule R_k. In the second column the firing strengths of $\mathbf{z}_2$, which belongs to C_2, are recorded. The fourth column of $\beta_{k,i}$ contains no elements as no sample falls in the according cluster C_4. The $z_{k,i}$ are finally obtained by normalizing $\beta_{k,1}$:

$$z_{k,i} = \frac{\beta_{k,i}}{\sum_{s=1}^{m} \beta_{k,s}} \qquad (15)$$

The tuning of the premises and consequences is done at a management station, e.g., a base station, where the training data has been collected. The estimated membership functions $\mu_{\tilde{A}_{k,i}}$ and the matrix $z_{k,i}$ are the only information which have to be downloaded onto the sensor nodes. Any subsequent classification of input $\mathbf{x} \in \Re^n$ can then be performed on the sensor nodes by applying Equation (11) .

4 Generation of Training and Test Data

In previous work [16] we proposed and investigated a distributed leader formation and data gathering algorithm (DELTA), which detects and tracks appearing events fast and efficiently. DELTA has been used to collect training and test data in this work too. For the current purpose the monitoring of static event sources with single-hop communication is sufficient, though. This can easily be satisfied with DELTA. The basic operation is depicted in Figure 3.

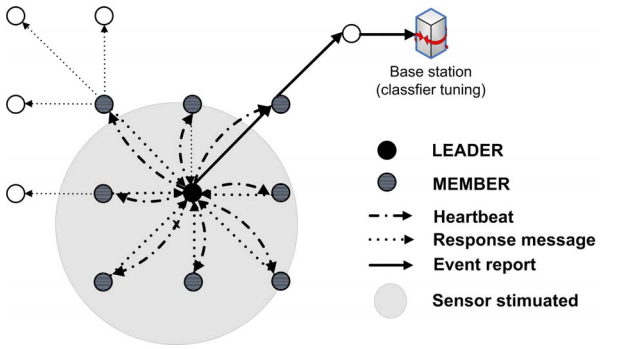

Figure 3. Event detection, group organization, localization, and reporting with DELTA.

The event observing region is represented as gray circle. Among the sensing nodes, a leader is elected (black node in Figure 3) based on the amplitude of the received signal strength(s). This is achieved with a timer which expires the sooner, the stronger the stimuli of the involved sensors on the node are. When elected, the leader immediately starts to broadcast messages (heartbeats) to inform its neighborhood about its state. Apart from the state dissemination, the heartbeat messages are also used to request data from the 1-hop neighbors of the leader. This data is required for the computation of event characteristics and classification. The response messages furthermore inform the two-hop neighborhood of the leader (white nodes in Figure 3) about its existence. Thus the probability of concurrently evolving leaders is decreased. To support tracking, the heartbeat/response message procedure is performed twice in a second.

The computation of the signal strength(s) emitted by an event is performed at the leader node based on the sensor readings collected in the neighborhood. Only the computed estimates are reported to the base station. In the following the computation of the emitted signal strength(s) is shortly summarized. Detailed information is provided in [16]. First, an adequate sensor model is required:

$$\rho_i = \frac{c}{\|\mathbf{x} - \xi_i\|^{\alpha}} + \omega \tag{16}$$

This model assumes an isotropic signal attenuation. The received signal strength ρ_i on a sensor node i, located at position ξ_i, decreases inversely to the distance to the event source, the location of which is $\mathbf{x}$. c is the amplitude of the emitted signal, α is the attenuation degree of signal c, ω is some additional white Gaussian noise, and $\|.\|$ is the Euclidean norm.

In the planar case there are n = 3 unknown variables in Equation (16). In space n is 4. Assuming the availability of k, with k $>$ n, sensor nodes, Equation (16) can be reformulated as a nonlinear least square objective function:

$$f(\mathbf{x}, c) = \sum_{i=1}^{k} \left(\rho_i - \frac{c}{\|\mathbf{x} - \xi_i\|^{\alpha}} \right)^2 \tag{17}$$

This function can be minimized by applying nonlinear optimization methods. So far, we tested the Nelder-Mead's Simplex Downhill (SD) [11] algorithm and the Conjugate Gradient descent method (CG) [11] in simulations. Based on its performance and simplicity, the SD method has been implemented and evaluated on real sensor nodes.

The TmoteSky [14] platform has been used for the current implementation. TmoteSky nodes consist of a microprocessor, some memory, an IEEE 802.15.4 compliant radio, and a number of sensors such as temperature, humidity, and light. Light can be measured with two sensors. The first light sensor measures only the photosynthetic active radiation (PAR), i.e., the visible light with a wavelength between 320 and 730 nm. The second light sensor measures the total solar radiation (TSR), including infrared, ranging from 320 to 1100 nm. In the current implementation these two sensors are used to classify different light sources. To generate the training and test sets five different light sources, bulbs of 25, 40, 60, 75, and 100 Watt, have been used. Thereby, the light bulbs were arranged in two different setups, which are depicted in Figure 4.

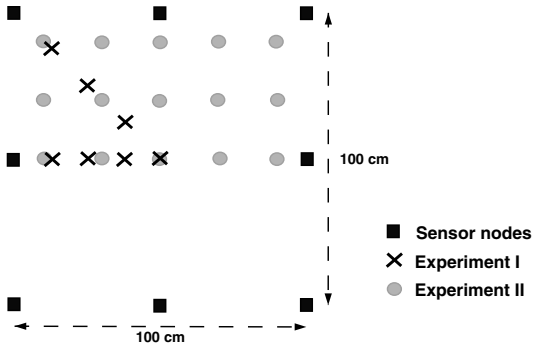

Figure 4. Test with 7 or 15 event positions.

In the first experiment the light sources have been placed at 7 specific locations, whereas in the second experiment 15 locations have been considered. Each experiment has been repeated 30 times. This results in training and test set sizes of 1050, respectively 2250, samples. According to DELTA

one of the sensor nodes in Figure 4 establishes as leader node and requests the sensor readings from its neighbors twice a second. Taking these readings as input, the SD algorithm computes the PAR and TSR values. The estimates are reported to the base station.

For the classification only the emitted light signal strengths are used. To get an idea of the samples collected at the base station, Table 1 shows the mean values (μ) and standard deviations (σ) of the training set of Experiment II.

Table 1. Emitted light signal strength.

	PAR		TSR	
Bulb	μ_1	σ_1	μ_2	σ_2
25 W	$3.24e^4$	$0.77e^4$	$3.16e^4$	$0.47e^4$
40 W	$5.72e^4$	$0.96e^4$	$5.23e^4$	$0.73e^4$
60 W	$9.61e^4$	$1.18e^4$	$7.94e^4$	$1.17e^4$
75 W	$12.07e^4$	$1.29e^4$	$9.54e^4$	$1.38e^4$
100 W	$17.47e^4$	$1.89e^4$	$12.09e^4$	$1.64e^4$

Considering μ_1 and σ_1, it seems as if the different light bulbs should be distinguishable in respect to the PAR value to some extent. The same is true for the TSR. Consequently, it should be possible to identify clusters (classes) representing the different event sources (light bulbs) from the training set. The data of Experiment I is very similar to the data of Experiment II and not shown here.

5 Classification Performance

The data from Experiment II, 15 event locations resulting in 2050 samples, has been used. Experiment I produced similar results, which are not shown. The classifiers are tuned with the training set and are then applied to the test set. The FLC classifier has been evaluated with both Gaussian and triangular membership functions. The FFNN has been implemented with 10 hidden neurons. Thus, the complexity of the FFNN is similar to that of the FLC.

The FLC and the FFNN filter events, which do not satisfy a given threshold T_μ. As a consequence some reporting delays are introduced. As mentioned in Section 4 classification is performed at the leader node every 0.5 s. In the current implementation, if T_μ is not satisfied after 30 computations, i.e., the reporting time expires 15 seconds, the classification is aborted. These settings are arbitrary and might be adapted based on the application.

Table 2 shows the performance of the different classifiers based on the classification error rate, i.e., the rate of wrong classifications (false positives) in dependence of T_μ.

All classifiers perform almost equally well if no filtering is applied ($T_\mu = 0$). Thus, a classification error-rate of approximately 10% can be achieved. With the FLC classifiers and the FFNN arbitrary classification error-rate constraints can be satisfied. If, for example, the classification error rate needs to be smaller than 5% (marked bold in Table 2), the FLC-Gaussian and the FFNN require a T_μ of 0.5, while FLC-Triangular requires a T_μ of 0.4. The Bayesian classifier does not support filtering, thus its classification error-rate remains constant.

Table 2. Error rates of the classifiers.

	Bayesian	FLC-Gauss	FLC-Tri	FFNN
T_μ	Error [%]	Error [%]	Error [%]	Error [%]
0	9.86	9.56	10.64	11.07
0.1		9.01	9.68	9.22
0.2		8.56	9.07	7.45
0.3		7.03	6.23	5.92
0.4		6.35	**4.6**	5.02
0.5		**4.85**	2.95	**3.34**
0.6		3.42	2.61	1.95
0.7		3.34	2.37	0.76
0.8		3.64	0	0.83
0.9		2.9		0

To evaluate the reporting delay five characteristic distribution parameters (in seconds) have been computed: The minimum and maximum, the median, and the lower and upper quartiles.

Table 3. Reporting delays of triangular FLC.

	Error rate	Latency percentiles p in [s]				
T_μ	[%]	Min	25-p	Median	75-p	Max
0	10.64	0.5	0.5	0.5	0.5	0.5
0.1	9.68	0.5	0.5	0.5	0.5	3.5
0.2	9.07	0.5	0.5	0.5	1	10
0.3	6.23	0.5	0.5	0.5	1	$> \mathbf{15}$
0.4	**4.6**	**0.5**	**0.5**	**1**	**6**	
0.5	2.95	0.5	0.5	1.5	7.5	
0.6	2.61	0.5	1	4.25	13.5	
0.7	2.37	0.5	2.5	10.5	> 15	
0.8	0	0.5	4.5	> 15		
0.9		0.5	> 15			

Table 3 shows the reporting latencies applying the triangular FLC classifier with varying T_μ. The minimum reporting delays to achieve an error-rate below 5% are marked bold. The results show that in 50% of all experiments (median) event reports are generated in 1 second. Accordingly, the classification needs to be performed only twice to generate an event report. In 75% of all experiments a report is sent within 6 seconds. At least in one experiment no report could be generated within 15 seconds. As expected, if T_μ is increased, the reporting delays become longer.

Table 4. Reporting delays of FFNN.

	Error rate	Latency percentiles p in [s]				
T_μ	[%]	Min	25-p	Median	75-p	Max
0	11.07	0.5	0.5	0.5	0.5	0.5
0.1	9.22	0.5	0.5	0.5	0.5	1
0.2	7.45	0.5	0.5	0.5	0.5	2.5
0.3	5.92	0.5	0.5	0.5	0.5	2.5
0.4	5.02	0.5	0.5	0.5	1	2.5
0.5	**3.34**	**0.5**	**0.5**	**0.5**	**1**	**4**
0.6	1.95	0.5	0.5	1	1.5	> 15
0.7	0.76	0.5	0.5	1	1.5	
0.8	0.83	0.5	1.5	1.5	3	
0.9	0	0.5	1.5	3	7	

The FFNN outperforms the FLC in terms of reporting delay (see Table 4). Considering an error-rate lower than 5%, in 50% of all experiments (median) a correct event report is sent immediately. Moreover, even in the worst case a report is reported within 4 seconds. This is because the neural network is able to model the distribution of the samples better than the FLC, which needs the abstraction of triangular or Gaussian distributions. On the other hand, the knowledge processing is hidden by the FFNN, making it more difficult to understand and configure. If the initial values are chosen poorly, the FFNN can perform quite worse than the FLC.

6 Conclusions and Future Work

The evaluation showed the trade-off between classification error-rate control and reporting latency. If the application requires a certain accuracy but can handle some delay, a neural network or a FLC classifier is a good choice. All three classifiers are performed at the leader node and require only local neighborhood information. Thus, the communication and computation overhead is kept small.

Till now only discrete events have been considered. In the future we aim at developing a classifier for intrusion detection in wireless sensor networks. Thereby, unauthorized office access will have to be detected. The associated classification problem includes the dimension of time, which will make high demands on the respective classifier.

References

[1] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3):293–313, 2002.

[2] M. Friedman and A. Kandel. *Introduction to pattern recognition: statistical, structural, neural, and fuzzy logic approaches*. World Scientific, 1999.

[3] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. A. Stankovic, T. Abdelzaher, and B. H. Krogh. Lightweight detection and classification for wireless sensor networks in realistic environments. In *Proc. of the 3rd international conference on Embedded networked sensor systems (SenSys '05)*, pages 205 – 217, San Diego, California, USA, November 2005.

[4] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proc. of the third international symposium on Information processing in sensor networks (IPSN '04)*, pages 1–10, Berkeley, California, USA, 2004.

[5] A. Kulakov and D. Davcev. Intelligent wireless sensor networks using fuzzyart neural-networks. In *Proc. of the 12th IEEE Symposium on Computers and Communications*, pages 569–574, Aveiro, Portugal, July 2007. IEEE.

[6] M. Kumar, L. Schwiebert, and M. Brockmeyer. Efficient data aggregation middleware for wireless sensor networks. In *Proc. of the First International Conference on Mobile, Ad-Hoc, and Sensor Systems (MASS '04)*, pages 579–581, Fort Lauderdale, Florida, USA, October 2004.

[7] L. I. Kuncheva. *Fuzzy Classifier Design*, volume 49 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, A. Springer-Verlag Company, 2000.

[8] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed. Detection, classification and tracking of targets. *IEEE Signal Processing Magazine*, 19(2):17–29, March 2002.

[9] L. Luo, T. F. Abdelzaher, T. He, and J. A. Stankovic. Envirosuite: An environmentally immersive programming framework for sensor networks. *ACM Transaction on Embedded Computing System (TECS)*, 5(3):543–576, 2006.

[10] M. Osborne, S. Roberts, A. Rogers, S. Ramchurn, and N. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In *Proc. of the 7th International Conference on Information Processing in Sensor Networks (IPSN '08)*, pages 109–120, Washington, DC, USA, 2008.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[12] M. Rabbat and R. Nowak. Distributed optimization in sensor networks. In *Proc. of the third international symposium on Information processing in sensor networks (IPSN '04)*, pages 20–27, Berkeley, California, USA, 2004. ACM.

[13] K. Römer. Discovery of frequent distributed event patterns in sensor networks. In *Proc. of the 5th European Conference on Wireless Sensor Networks (EWSN '08)*, pages 106–124, Bologna, Italy, January 2008.

[14] Sentilla. Pervasive computing solutions, July 2008.

[15] X. Sheng and Y.-H. Hu. Maximum likelihood multiple-source localization using acoustic energy measurements with wireless sensor networks. *IEEE Transactions on Signal Processing*, 53(1):44– 53, January 2005.

[16] M. Wälchli, S. Bissig, M. Meer, and T. Braun. Distributed event tracking and classification in wireless sensor networks. *Journal of Internet Engineering*, 2(1):117–126, 2008.

[17] T.-Y. Wang, Y. S. Han, P. K. Varshney, and P.-N. Chen. Distributed fault-tolerant classification in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):724–734, 2005.