Computer Communications xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Hierarchical distributed data classification in wireless sensor networks ☆

Xu Cheng, Ji Xu, Jian Pei, Jiangchuan Liu *

School of Computing Science, Simon Fraser University Burnaby, British Columbia, Canada V5A 1S6

ARTICLE INFO

Article history:
Received 24 September 2009
Received in revised form 18 January 2010
Accepted 30 January 2010
Available online xxxx

Keywords:
Wireless sensor network
Classification
Decision tree
C4.5

ABSTRACT

Wireless sensor networks promise an unprecedented opportunity to monitor physical environments via inexpensive wireless embedded devices. Given the sheer amount of sensed data, efficient classification of them becomes a critical task in many sensor network applications. The large scale and the stringent energy constraints of such networks however challenge the conventional classification techniques that demand enormous storage space and centralized computation.

In this paper, we propose a novel decision-tree-based hierarchical distributed classification approach, in which local classifiers are built by individual sensors and merged along the routing path forming a spanning tree. The classifiers are iteratively enhanced by combining strategically generated pseudo data and new local data, eventually converging to a global classifier for the whole network. We also introduce some control factors to facilitate the effectiveness of our approach.

Through extensive simulations, we study the impact of the introduced control factors, and demonstrate that our approach maintains high classification accuracy with very low storage and communication overhead. The approach also addresses a critical issue of heterogeneous data distribution among the sensors.

1. Introduction

The recent advances in transceiver and embedded hardware designs have made massive production of inexpensive wireless sensors possible. A wireless sensor network (WSN) consists of a number of sensor nodes (few tens to thousands) storing, processing and relaying the sensed data, often to a base station for further computation [1,2]. Wireless sensor networks can be used in many applications, such as wildlife monitoring [3], military target tracking and surveillance [4], hazardous environment exploration [5], and natural disaster relief [6]. Given the huge amount of sensed data, classifying them becomes a critical task in many of these applications.

As an example, for wildlife monitoring, the sensor nodes continuously sense the physical phenomena such as temperature, humidity and sunlight, and meanwhile may also count the number of animals. The number reflects the suitability of the current environment for the animals, for example, if it is greater than a threshold, the environment is classified as suitable, and otherwise not. After learning the relation between the physical phenomena and the classes from such *training data*, we may later determine the suitability of the inquired environment from the external source. These inquires are the *unseen data* which only have the physical phenomena but do not have the class label. The sensor data collection and object count have been extensively studied in the literature, e.g., the Great Duck Island Project [3], yet efficient classification for wireless sensor networks has not been well addressed.

Classification is typically done in two steps: first, a classifier is constructed to summarize a set of predetermined classes, by learning from a set of training data; then, the classifier is used to determine the classes of newly arrived data. Within this framework, there have been significant efforts in improving its speed and accuracy, most of which assume centralized storage and computation. The wireless sensor networks however pose a series of new challenges, particularly for the first step. First, the number of sensor nodes is huge, but each of them has only limited storage that can hardly accommodate all the training data of the whole network. Second, the sensor nodes are generally powered by non-rechargeable batteries, and energy efficiency is thus of paramount importance, which makes the straightforward solution of sending all the training data to the powerful base station quite inefficient [7,8]. In short, conventional centralized solution is not directly applicable in this new type of network environment.

To address the above challenges, in this paper, we present a novel decision-tree-based hierarchical distributed classification in energy-constrained sensor networks. Our approach organizes the sensor nodes and performs classification in a localized, iterative and bottom-up manner, utilizing a decision tree method, in particular, an enhanced C4.5 algorithm. Starting from each leaf node, a classifier is built based on its local training data. An upstream node,

☆ Extended version from IEEE MASS'09; Invited for fast-track journal publication.

* Corresponding author. Tel.: +1 778 782 4336; fax: +1 778 782 3045.
E-mail addresses: xuc@cs.sfu.ca (X. Cheng), jix@cs.sfu.ca (J. Xu), jpei@cs.sfu.ca (J. Pei), jcliu@cs.sfu.ca (J. Liu).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.01.027

upon receiving the classifiers from its children, will use them with its own data to build a new classifier. These local classifiers will be iteratively enhanced from bottom to top and finally reach the base station, making a global classifier for all the data distributed across the sensor nodes. Since only the classifiers, instead of the sensed data, will be forwarded upstream, the energy consumption for transmission can be significantly reduced.

Our analysis suggests that, given certain amount of sensor nodes of the same locations, a high tree will save more energy but a wide tree will achieve higher accuracy. We propose a spanning tree constructing approach, in which each node selects a portion of the candidate children as the children to control the shape of the spanning tree. Another key difficulty lies in training a new classifier from a mix of downstream classifiers and the local training dataset, which cannot be directly accomplished by the existing learning algorithms that work on dataset only. We address this problem by generating a *pseudo training dataset* from each downstream classifier. We develop a smart generation algorithm, which ensures that the pseudo data closely reflect the characteristics of the original local data. We also introduce a control parameter that adaptively balances the recovery quality and the amount of the data. Through extensive simulations, we demonstrate that our approach maintains high classification accuracy, with very low storage and communication overhead.

We also notice that, in practice, the data distribution across different sensor nodes is not necessarily homogeneous. For example, depending on the location, the data sensed by one node may always have low temperature and low humidity, and the data sensed by another node at a different location may always have high temperature and high humidity. We show strong evidence that such heterogeneity can easily lead to misclassification, and we propose an enhanced C4.5 algorithm to mitigate its impact. To our knowledge, it is the first solution addressing this distribution issue.

The remaining part of the paper is organized as follows. Section 2 gives a brief introduction of classification. Section 3 describes our approach, including constructing spanning tree, building local classifier, generating pseudo data, and building the global classifier hierarchically. We evaluate our approach and present the results in Section 4. Section 5 reviews the related works in the literature. Finally, Section 6 concludes the paper.

2. Classification basics

2.1. Classification

Classification, which is the task of assigning objects to one of several predefined categories, is a pervasive problem that encompasses many diverse applications [9,10]. Data classification is a two-step process. In the first step, a model is built describing a predetermined set of data classes or concepts, and the model is constructed by analyzing database *samples* described by attributes. Each sample is assumed to belong to a predefined class, as determined by one of the attributes, called the class label attribute. The samples analyzed to build the model collectively form the training data set. Since the class label of each sample is provided, this step is also known as *supervised learning*. Typically, the learned model is represented in the form of decision trees, classification rules, or mathematical formula.

In the second step, the model is used for classification. First, the predictive accuracy of the model (or *classifier*) is estimated. A simple way is to use a *test set* of class-labeled samples, which are randomly selected and are independent of the training samples. The accuracy of a model on a given test set is the percentage of test set samples that are correctly classified by the model. If the accuracy of the model is considered acceptable, the model can be used to classify future data of which the class label is not known.

2.2. Decision tree

Decision tree is one of the most important models for classification, and also serves as the foundation for our classification method. A decision tree is a mapping from observations about an item to conclusions about its target value. A decision tree has three types of nodes: (1) a root node that has no incoming edges, (2) internal nodes, each of which has one incoming edge and more than one outgoing edges, (3) leaf nodes, each of which has one incoming edge and no outgoing edge. In a decision tree, each leaf node is assigned a class label. The root and internal nodes, contain attribute test conditions to separate data that have different characteristics.

Classifying a test data is straightforward once a decision tree has been constructed. Starting from the root node, the test condition is applied to the data and follows the appropriate branch based on the outcome of the test. This will lead to either another internal node, for which a new test condition is applied, or a leaf node. The class label associated with the leaf node is then assigned to the record. Fig. 1 shows a decision tree example. For instance, it indicates that if the temperature is between 15 and 30 and the sunlight is weak, the environment is classified as suitable (Yes) for animals.

The challenge lies in constructing the decision tree is how to find the best attribute to split the sample data. The following describes two common criteria.

[*Information gain*:] The information gain is the simplest criterion. It uses the entropy measure, which can be calculated as

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i,$$

where S is the dataset, c is the number of classes and p_i is the proportion of each class. The information gain is then calculated as

$$Gain(S,A) = Entropy(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropy(S_v),$$

where $V(A)$ is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

[*Gain ratio*:] There exists natural bias in information gain, as it favors attributes with many values. For example in weather forecast, the attribute "data" may have the highest information gain, but it will lead to a very broad decision tree of depth one and is inapplicable to any future data. There are some advanced criteria, such as gain ratio, which penalizes attributes by incorporating split information

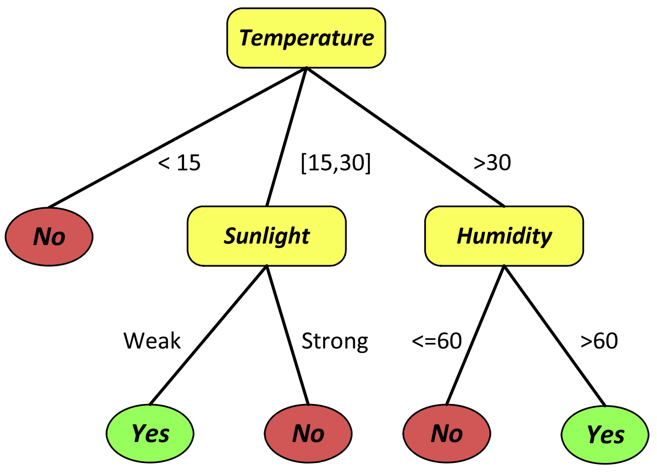

Fig. 1. An example of a decision tree.

$$Split\ Information(S,A) = -\sum_{i=1}^{c}\frac{|S_v|}{|S|}\log_2\frac{|S_v|}{|S|}.$$

This information is sensitive to how broadly and uniformly the attribute splits the data. The gain ratio is calculated as

$$Gain\ Ratio(S,A) = \frac{Information\ Gain(S,A)}{Split\ Information(S,A)}.$$

Note that this ratio is not defined when the split information is zero, and the ratio may tend to favor attributes for which the split information is very small. Consequently, to deal with this situation, the information gain is calculated for all attributes, and then the method only selects the best one.

There are various methods to build a decision tree. Among them, *ID3*, proposed by Quinlan [11], is the simplest and most famous one. The ID3 algorithm recursively constructs a tree in a top-down divide-and-conquer manner. It uses information gain as the measure to determine the best attribute, and then creates a node for each possible attribute value, and partitions the training data into descendant nodes. There are three conditions to stop the recursion: (1) all samples at a given node belong to the same class; (2) no attribute remains for further partitioning; (3) there is no sample at the node.

Later, Quinlan proposed an extension of ID3 algorithm, C4.5 [12]. It performs similarly to ID3 except using gain ratio to determine the best attribute. C4.5 algorithm also makes some improvements to ID3, including that it can handle numerical attributes by creating a threshold and splitting the data into those whose attribute value is above the threshold and those that are less than or equal to it. C4.5 can also prune the decision tree after creation, which reduces the size of the tree.

However, both original ID3 and C4.5 algorithms cannot be used to directly combine the local classifiers, nor do they preserve the data attribute distribution to support pseudo data recovery. In this paper, we will present a substantially enhanced version to address these challenges.

3. Decision-tree-based hierarchical distributed classification

In this section, we first present the system overview, and then introduce our classification approach in detail, showing how to construct the spanning tree, how to build the local decision tree, how to generate the pseudo data, and how to build the classifier in a bottom-up manner. We discuss the accuracy and the energy consumption afterwards.

3.1. System overview

We consider N sensor nodes $n_1, n_2, \ldots, n_N$ distributed in a field. Each node covers an area of the field and is responsible for collecting data within the area. The data reporting follows a spanning tree rooted at the base station n_0. The routing protocol for forming spanning tree will be presented in Section 3.2.

Each sensor node n_i first collects its local training data D_i. If node n_i is a leaf node, it builds a classifier C_i by a learning algorithm $\aleph$, which we will illustrate in Section 3.3. The node then sends C_i to its parent node, say n_j. We use a decision tree to represent the classifier,[1] which, compared to the original data, is of a much smaller size.

[1] We will use "classifier" and "decision tree" interchangeably throughout this paper provided the context is clear. Also note that, the decision tree and the spanning tree are two concepts, as the former is the classifier and the latter is the system structure.

An upstream node n_j, upon receiving the classifiers from its children, combines the children's classifiers with its local training data D_j to build an enhanced classifier C_j. These local classifiers will be iteratively enhanced from bottom to top and finally reach the base station, making a global classifier for all the data distributed across the sensors. Since only the classifiers will be forwarded upstream, the energy consumption for transmission can be significantly reduced. The sensor nodes may continuously sense new data and forward to upstream.

The challenge here lies in training the enhanced classifier from a mix of downstream classifiers and the local dataset, which cannot be directly accomplished by the existing training algorithms that work on dataset only. To address this problem, a pseudo training dataset will be generated from each downstream classifier. For each child node n_i, node n_j will generate a set of pseudo training data D_i' from the classifier C_i, and then combine all these data with its own training data to build the enhanced classifier. Obviously, the pseudo data, recovered from classifiers, should closely reflect the characteristics of the original local data, e.g., the distribution of different classes and the attribute values. The amount of the pseudo data is also an important concern given that a sensor node generally has a limited memory. We will address these detailed issues in Section 3.4.

We list the major notations in Table 1. Also note that, in this paper, we do not consider issues like node failure or packet loss, which have been extensively addressed in the literature [7,8,13].

3.2. Constructing spanning tree

Before we present the approach to construct the spanning tree, we first discuss how the shape of the tree affects the result of the classification; specifically, how the height and width of the tree affect the energy consumption and the accuracy of the classifier.

Given a certain amount of sensor nodes, it is intuitive that if the tree height is small, the tree width is large, and vice versa. Suppose in Fig. 2, node 0 is the base station and there are n other sensor nodes in one line with equal interval, and the distance between the base station and the last one is L. We assume there are two kinds of spanning tree, one is that each sensor node is the child of the other one that is closer to the base station, representing a high tree T_1, and the other is that all these sensor nodes are the child of the base station, representing a wide tree T_2.

In general, the energy consumption of computation is significantly smaller than that of transmission, and the transmission energy consumption is proportional to the transmitted data size and square of the distance [8]. In T_1, the energy consumption is $\sum_{i=1}^{n} s\cdot\frac{L^2}{n}$, where s is the data size transmitted by a sensor node. While in T_2, the energy consumption is $\sum_{i=1}^{n} s\cdot\frac{i\cdot L^2}{n}$, which is clearly larger than that of T_1. Therefore, to save more energy, a high tree is demanded. On the other hand, noise in the classification is inevitable, and the noise will be accumulated along the spanning tree. Hence the larger the height is, the less accurate the classifier will be, and our evaluation demonstrates this. Therefore, we need an approach to control the shape of the spanning tree during its construction.

Table 1
List of notations

Notation	Explanation
N	The number of sensor nodes, excluding the base station
n_i	Sensor node $(i = 1, 2, \ldots, N)$
n_0	Base station
D_i	The training data collected by node n_i
D_i'	The pseudo data generated from classifier C_i
C_i	The local classifier built by node n_i
C_0	The global classifier built at the base station

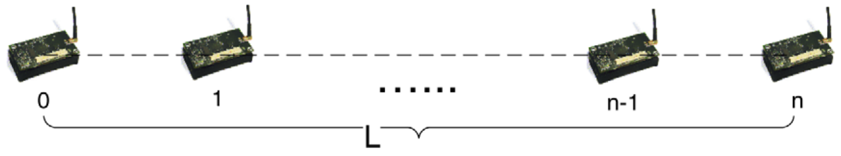

Fig. 2. Analysis of the spanning tree height.

In our spanning tree constructing approach, the base station first broadcasts *tree construction message* containing the source (base station) and the depth (0). Upon receiving the message, the sensor nodes within the range of the base station send response messages back to the base station, and these nodes are candidate children. The base station selects all or some of these sensor nodes as its children, and the number of children will affect the shape of the spanning tree. A random selection is simple but effective, which avoids the issue of load balance. The base station then sends confirmation messages to the all the candidate nodes, informing them whether or not they have been selected as the children. The selected nodes are the depth 1 nodes.

Then iteratively, the sensor nodes broadcast tree construction message to select their children, until all the sensor nodes have a parent node, and this forms the entire spanning tree. Note that, the sensor node always prefers a parent with a smaller depth, to achieve a shorter distance. However, that parent might not necessarily select the node as child, hence the process of constructing spanning tree is in a top-down manner. The advantage of our spanning tree constructing approach is that, the tree width can be controlled by setting the maximum number of the selected children from those who respond the message, and from another perspective, this also controls the height of the spanning tree.

3.3. Building decision tree

In our system, the local decision trees are built by the widely-used C4.5 algorithm [12]. The basic C4.5 algorithm, however, does not keep the information about the attribute distribution and the amount of the original training data, preventing pseudo data recovering from a classifier.

To solve this problem, we let each leaf node record the count of each class (i.e., the number of positive and negative labels in this application scenario). Therefore, we have the knowledge about the amount of the samples for building each branch of the decision tree, and thus we can make the distribution of the generated pseudo data resemble the original ones.

Moreover, in the basic C4.5 algorithm, if all the samples belong to the same class, the recursion stops (e.g., when temperature is low in Fig. 1). Hence, the information of other attributes will be missing, which can cause problems with heterogeneous data distribution across different sensor nodes. For example, if all the training data sensed by a sensor node are below 10° in temperature and below 20% in humidity, and the class labels are all negative, using the basic C4.5 algorithm, only one attribute, say temperature, will appear in the decision tree, and the information of humidity is completely missing. This will likely lead to a set of pseudo data generated with humidity uniformly distributed from 0% to 99%, which is clearly not the case for the original data.

Therefore, for the stop condition of the recursion, we eliminate the first one (referred to Section 2) in our enhanced C4.5 algorithm. The new stop condition thus becomes when no attribute remains for further partitioning or when there is no sample.

In Algorithms 1 and 2 below, we describe the enhanced C4.5 algorithm for building the decision tree. Note that a brief illustration of the key steps of the basic C4.5 is in Section 2, and more details can be found in Quinlan's work [12].

Algorithm 1. LearningAlgorithm (D)

```
Require: training dataset $D$
  call EnhancedC45 ($root$, $D$, 0)
  prune decision tree $root$
  return $root$
```

Algorithm 2. EnhancedC45 (pt, D, $depth$)

```
Require: pointer to the decision tree node $pt$, training dataset $D$,
  depth of the decision tree node $depth$
  if number of data $|D| = 0$ then
    return
  end if
  if $depth$ = number of attributes of the training data then
    get the class label with the most count in $D \rightarrow$ attribute of $pt$
    record the count of each class label for $pt$
    return
  end if
  calculate gain ratio for each attribute
  get the attribute with the greatest gain ratio $\rightarrow target\_attribute$
  if $target\_attribute$ is category attribute then
    for all $target\_attribute$ value branches do
      add a child $pt'$
      set the value of the branch $v_i$
      partition $D \rightarrow D'$ with value $v_i$
      call EnhancedC45 ($pt'$, $D'$, $depth + 1$)
    end for
  else {//numerical category}
    for all $target\_attribute$ value branches do
      add a child $pt'$
      set the value range of the branch $[v_i, v_{i+1})$
      partition $D \rightarrow D'$ satisfying the value range $[v_i, v_{i+1})$
      call EnhancedC45 ($pt'$, $D'$, $depth + 1$)
    end for
  end if
```

3.4. Generating pseudo data

The pseudo data generation is one of the most important steps in our framework. A critical challenge here is to generate data that are as close to the original data as possible. In particular, the distribution of each attribute should closely resemble that of the original data.

Another issue is how many pseudo data should be generated. Intuitively, the fewer data we generate from the child nodes, the less weight they have. Since data from one node should not be considered less important than those from another, we need to generate the same amount of the pseudo data as the original data. Therefore, a sensor node close to the base station has to generate a huge amount of pseudo data, i.e., the same amount of the original data at all its descendants (not only the immediate children). This is often impossible given the limited memory of the embedded sensor nodes. To this end, we introduce a *preservation factor*, ranging from 0 to 1 (the base station always has a factor of 1), to control the amount of the generated pseudo data.

Algorithm 3. GeneratePseudoData (C, pf)

```
Require: decision tree received from one child $C$, preservation
  factor $pf$
for all leaf nodes $node$ of decision tree $C$ do
    get rule $R$ of $node$
    get class label counts $\rightarrow c_1, c_2, \ldots, c_L$
    randomly generate $pf \cdot \sum_{i=1}^{L} c_i$ data satisfying $R$
    assign class label $l_k (k = 1, \ldots, L)$, to the data with probability
  as the proportion of the class label $c_k / \sum_{i=1}^{L} c_i$
    add these data to pseudo data set $D'$
end for
return pseudo data set $D'$
```

Algorithm 3 summarizes our method to generate the pseudo data. For illustration, suppose the decision tree's leaf node represents a rule of when temperature is between 10 and 20, humidity is between 20 and 40, and sunlight is *normal*, there are 10 positive class labels and 90 negative ones. As such, the class label is negative. Assuming the preservation factor is set to 0.6, we then randomly generate $(10 + 90) \times 0.6 = 60$ data that satisfy the attribute requirement. It follows that each data has a probability $10/100 = 0.1$ to be assigned a class label as positive and 0.9 to be negative.

The original data are partitioned to each decision tree leaf node, and each set of generated data resembles part of the original data. Therefore, the combined pseudo data will largely reflect the characteristics the original training data. We will closely examine the effectiveness of our method as well as the impact of the preservation factor in Section 4.

3.5. Hierarchical classification

As mentioned above, we let the sensor nodes in the network be organized by a spanning tree, as illustrated in Section 3.2, and Fig. 3 shows an example. A leaf node builds the decision tree with the local sensed training data and sends the decision tree to the parent. The intermediate node periodically checks if there is any new classifier from children. If yes, it will generate a set of pseudo data for each new classifier, and combines them with its local data. There are two situations here. (1) If the node has never built any classifier, which indicates that it has never received any classifier from its children, it will combine the generated pseudo data with its local sensed data and performs the learning algorithm. (2) If the node has once built a classifier, the previous received classifiers may have already been discarded (due to the memory constraint). The node will then generate a set of pseudo data for its local classifier with the preservation factor being 1, and combines it with the other pseudo datasets to build the new decision tree. We prefer a pull-based method, because if we utilize a push-based method, an update of the leaf node will trigger a series of re-computations and transmissions all the way to the root, consuming great energy. In fact, the updated classifier is sent to the parent node immediately after the update, but the parent node might do nothing until the next period, and note that, it costs nothing for the parent node to check if there is any new classifier.

The base station will build the global classifier. Initially, it waits until receiving all the classifiers from its children, and then generates pseudo data for each, and combines them to build the decision tree. Since the base station in general is a more powerful node, it could store all the classifiers from its immediate children. Therefore, when one of them updates the classifier, the base station discards the old one and re-performs the operations as above.

In Algorithm 4, we summarize the detailed procedure of the bottom-up classification.

Algorithm 4. BottomUpClassification ()

```
if node is leaf then
    periodically collect data $D$
    $C \leftarrow \text{LearningAlgorithm}(D)$
    send $C$ to its parent
else if node is not base station then
    periodically collect data $D$
    for all new classifiers $C_k$ from child $k$ do
        $D'_k \leftarrow \text{GeneratePseudoData}(C_k, \alpha)$
    end for
    $C \leftarrow \text{LearningAlgorithm}(D \cup (\bigcup D'_k))$
    send $C$ to its parent
else {// base station}
    if no classifier has been built then
        for all classifiers $C_j$ from child $j$ do
            $D'_k \leftarrow \text{GeneratePseudoData}(C_k, 1)$
        end for
        $C \leftarrow \text{LearningAlgorithm}(\bigcup D'_k)$
    else if receive new classifier $C_k$ from child $k$ then
        replace old classifier of child $k$ with $C_k$
        for all classifiers $C_k$ from child $k$ then
            $D'_k \leftarrow \text{GeneratePseudoData}(C_k, 1)$
        end for
        $T \leftarrow \text{LearningAlgorithm}(\bigcup D'_k)$
    end if
end if
```

3.6. Further discussion

Our approach utilizes the C4.5 as the basis for classification, so it inherits the effectiveness and efficiency of C4.5 when building local classifiers [12]. To make it fit our application scenario better, we define that all the leaf nodes in the decision tree (classifier) have the same depth as the number of the attributes, so that we can keep all the attribute information when abstracting the rule from each branch. Thanks to the modification, we can generate the pseudo data that are very similar to the original data. Apparently, this modification will increase the size of the decision tree, however, such increase is acceptable and it noticeably increases the classification accuracy, as will be validated in our performance evaluation.

In order to keep the high accuracy and achieve our goals of saving energy and storage space, we have introduced two parameters in our solution. One is the *class count*. It not only records the count but also indicates the distribution of all the class labels in the original dataset when we build the decision tree. In other words, we keep the "noise" information because the "noise" may be important in some cases, in particular, the heterogeneous data distribution. For example, suppose one decision tree branch has the negative label because the numbers of positive and negative training data satisfying the constraint are 1 and 9, while another decision tree branch has the positive label on the same attribute constraint because the counts are 99 and 1. Without recording the class label counts, we have no idea about which one is more accurate, and we will likely treat them equally. In fact, combining the two training dataset, we will obtain the positive label with the class counts 100 versus 10. The problem is particularly severe when the training dataset have the heterogeneous distribution, which critically demands the recording of the class count.

The other parameter is the preservation factor, which determines the amount of the pseudo data to generate. We introduce this parameter due to the limited memory of the sensor node. The smaller the preservation factor, the more dominant the local training data are. For example, suppose a node has four children, each having 200 training data, and the node itself also has 200 training data. If the preservation factor is set to 1.0, the node will

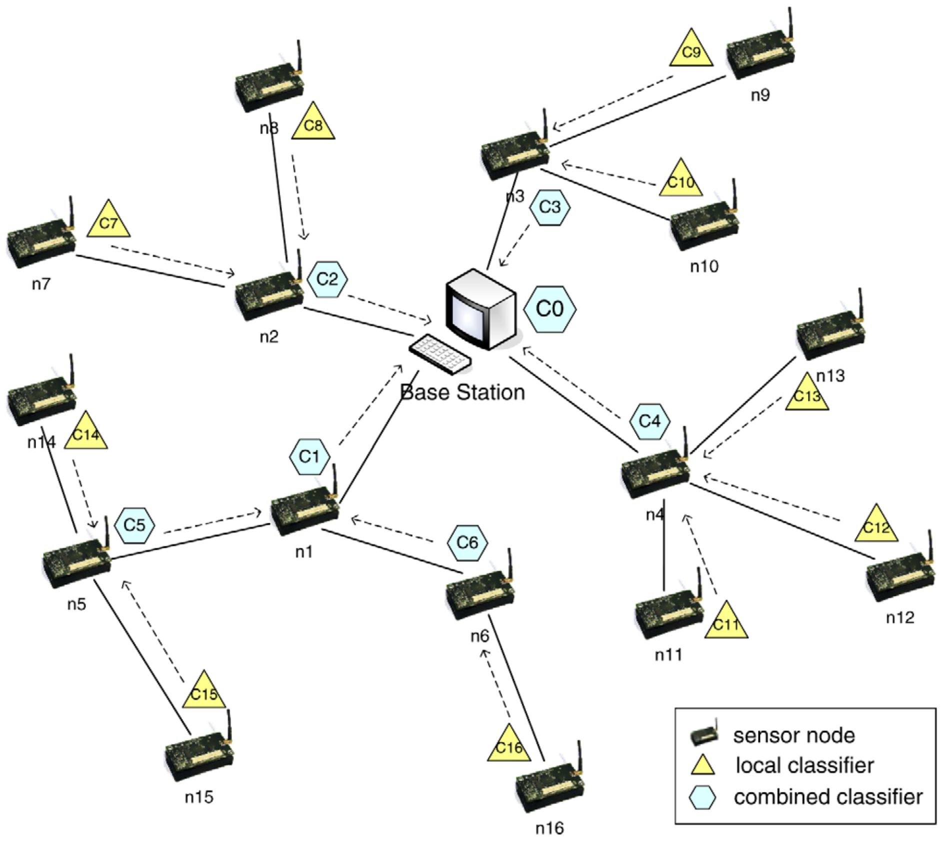

Fig. 3. The structure of the sensor network.

learn from 1000 data, in which 20% are its local data. If the preservation factor is set to 0.1, the node will learn from 280 data, in which 71% are its local data. Intuitively, if the pseudo data can represent the original data very well, the greater the preservation factor, the more accurate the classifier is, because every area should be treated equally. Otherwise, the greater the preservation factor, the more noise it will make, thus decreasing the accuracy. Therefore, the representativeness of the pseudo data of the original data is crucial in our solution. In the next section, we will closely examine its impact to the classification accuracy.

To extend our previous work [14], we adopt C4.5 instead of ID3 as the basis for building the decision tree. Two important improvements of C4.5 over ID3 are, C4.5 can deal with numerical attribute and can prune the tree after creation. To explore the two advantages, during creating the decision tree, we can set the attribute granularity small enough, which however increases the tree size; then after creating the tree, the sibling nodes can be merged if they have the same distribution. In the next section, we will demonstrate that the new method can save more energy.

4. Performance evaluation

4.1. Configuration and dataset

We evaluate our framework with our customized simulator. We consider a square field consisting of $m \times m$ randomly deployed sensor nodes, with the base station being located in the center. We set m as 7 (totaling 49 nodes), and we control the spanning tree constructing approach to get the spanning trees that have heights of 3, 4 and 5, respectively. Fig. 4 shows an example with height 4. We have tested the topology with larger height, and the results show the same trend as the smaller height, thus we only provide the results of height being 3, 4 and 5. We have also tested different values of m (up to 20) and obtained the same results, which indicates that the performance of the algorithm is mainly affected by the height of the spanning tree, while not the node populations. This is because the algorithm is distributed and localized. For the ease of analysis, we take $m = 7$ as the experiment example.

The data has three attributes and a class label. The three attributes are temperature, humidity and sunlight. The first two are numerical attributes, ranging from 0 to 49 and from 0 to 99, and the last one is a categorical attribute, having values *weak*, *normal* and *strong*. Since we utilize C4.5 algorithm which can deal with numerical attribute, it improves our previous work [14]. The class label is either *positive* or *negative*, i.e., indicating whether or not the environment is suitable for the animal.

We randomly generate data having the temperature between 0 and 49, humidity between 0 and 99, and sunlight being 0, 1 or 2; we manually define some rules to assign each data a class label. We also consider noise and thus add a factor ε, which indicates the data has the probability of ε to be the other class label determined by the rules. Note that this noise has nothing to do with the transmission error, and we suppose the underlying protocol can successfully handle the error. This noise indicates the inaccuracy of a generative model representing the dataset, and we study two typical cases in the following.

We generate 10 training datasets, each having $200 \cdot (m^2 - 1)$ data. Among the 10 datasets, five of them have $\varepsilon = 1\%$ noise, and five of them have $\varepsilon = 10\%$ noise. For each dataset, we make it into two versions, one is called *heterogeneous data* and the other is called *homogeneous data*. In the heterogeneous data set, the data distribution depends on the location of the sensor nodes, while the homogeneous data is independent on the location, and is randomly and uniformly distributed across sensor nodes.

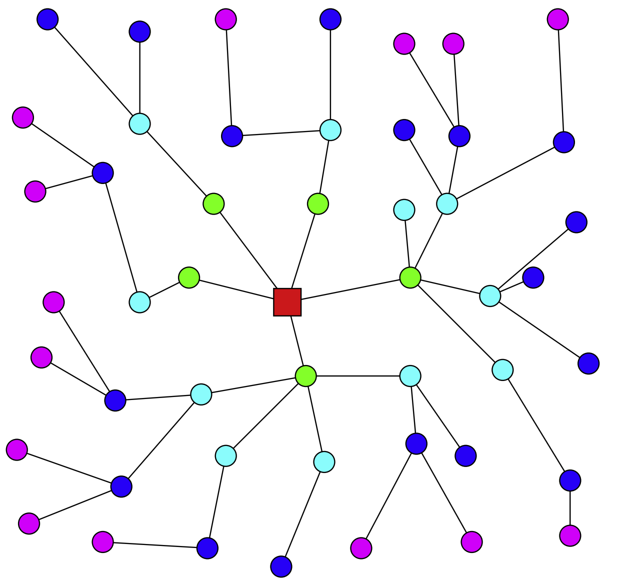
Fig. 4. An example of topology of 49 nodes and height being 4 (different colors show represent different depths).

For homogeneous data, we just randomly divide and assign all the training data to the $(m^2 - 1)$ sensor nodes as the local training data, thus each sensor node has 200 training data. For heterogeneous data, we assume that the temperature increases from left to right, and the humidity increases from bottom to top in the sensor field; the attribute of sunlight is uniformly distributed. For example, a node in the bottom right corner is supposed to have the data with high temperature around 45, low humidity around 10, and a node in the top left has the data with low temperature around 5, high humidity around 90. For each data in one dataset, we first calculate its coordinates according to the above, and then assign the data to the sensor node of that location if the training set of the node is not full (200 data). If that node is full, we then assign this data to another random node. For each training dataset, we perform different simulations that have different data distributions. For example, one is described as above, and another one is that temperature increases from bottom to top, humidity increases from right to left, and so forth.

In our experiments, we generate 10 test datasets, each having 2000 data. Among the 10 datasets, five of them have $\varepsilon = 1\%$ noise, and five of them have $\varepsilon = 10\%$ noise. If we use the training data with $\varepsilon = 1\%$ noise to learn, we use the test data with $\varepsilon = 1\%$ noise to test (same as the data with $\varepsilon = 10\%$ noise).

4.2. Baseline for comparison

We have also implemented an *ensemble method* [15] for the baseline comparison. The ensemble method constructs a set of base classifiers and take a majority voting on predictions in classification. The method can significantly improve the accuracy of prediction, because if the base classifiers are independent, then the ensemble makes a wrong prediction only if more than half of the base classifiers are wrong. For example, suppose there are two classes and each base classifier has an error rate of 30%. With 15 base classifiers, the error rate will be $\sum_{i=8}^{15} \binom{15}{i} 0.3^i 0.7^{15-i} = 0.05$ only.

The ensemble method has been widely adopted in distributed classification [16]. In our evaluation, we customize the ensemble method to our application scenario. Specifically, all the nodes learn from their local training data to build the classifiers, and send the classifiers to the base station through multi-hop routing. The base station then conducts the second step of the classification, in particular, the base station tests the unseen data with all the classifiers and takes a majority voting to get the final decision. Obviously, sending all the local classifiers to the base station consumes more energy than only sending the local classifier to the parents. Moreover, the ensemble method does not accommodate heterogeneous data, thus the accuracy of classification can be low, as will be shown later.

4.3. Impact of preservation factor, noise and height for heterogeneous data

We first examine our algorithm with different preservation factors, noises and heights for the heterogeneous data. We plot the results in Fig. 5. The x axis is the preservation factor ranging from 0.1 to 1.0, and we also evaluate the best-possible accuracy of learning from the entire dataset (assuming one sensor node collects all the data and builds the classifier), referred to as “A”, and accuracy of the ensemble method, referred to as “E”.

From the figure, we find that the preservation factor does affect the accuracy, especially when it is small, the height is big, and the noise is large. When the noise is very small (1%), the preservation factor does not affect the accuracy much when the factor is larger than 0.4, regardless of the height. When the noise becomes greater (10%), the preservation factor should be at least 0.7, so as not to decrease the accuracy much. When the preservation factor is very small, the larger the height is, the less accuracy it achieves. We also find that when the preservation factor is large enough, the accuracy is almost the same as that of learning from the entire training data, i.e., the practically optimal accuracy.

The figure indicates that our mechanism to generate pseudo data works quite well, in particular, when the factor is large enough, the pseudo data can well represent the original data. When the preservation factor is small, the accuracy becomes relatively lower. The reason is that, from the whole system's view, the areas that are close to the base station generally dominates, but as mentioned before, we should consider different areas equally. Moreover, when the height is higher, the noise will be accumulated, hence the accuracy will be reduced.

Comparing with the ensemble method, when the preservation factor is large enough, our approach achieves much higher accuracy, and when the noise is greater, the difference is even larger. This is because in the ensemble method, for heterogeneous data, each sensor node only learns a part of the data (e.g., low tempera-

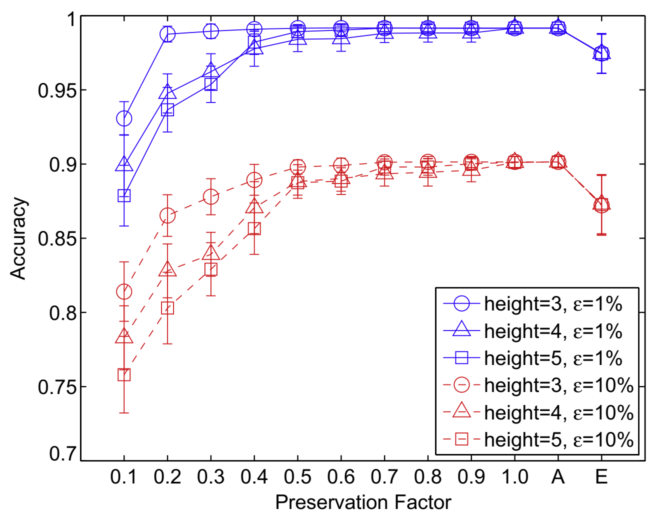

Fig. 5. Comparison of accuracy for different preservation factor, noise and height with heterogeneous data.

ture and low humidity). In other words, in the ensemble method, only a few classifiers are responsible for a certain test data. If a given unlabeled data has the attribute value that is much different from its training data, it probably needs to randomly guess a class label, which will greatly decrease the accuracy.

4.4. Comparison of enhanced and basic C4.5 algorithm

We next compare our enhanced C4.5 algorithm with the basic C4.5 algorithm, as well as the enhanced ID3 algorithm proposed in our previous work [14]. We clarify that we modify the basic C4.5 because it is not suitable for generating pseudo data in our approach for the heterogeneous data distribution. We again use the heterogeneous data with different noises, and plot the results in Figs. 6 and 7, respectively.

We find that the enhanced C4.5 algorithm is not noticeably affected by the preservation factor when the factor is large enough, nor the height. On the other hand, the accuracy of basic C4.5 algorithm is much lower than ours, and is particularly lower when the preservation factor is smaller and the height is larger.

The difference is because when all the samples belong to the same class label, the basic algorithm stops the recursion, while our enhanced version continues. As mentioned before, in heterogeneous data distribution, it is possible that all the data from one node are below 10 in temperature and below 20 in humidity, and all the labels are negative; if we utilize the basic ID3, the decision tree is likely to contain only one attribute, say temperature, with the information of humidity being completely missing. As such, when generating the pseudo data, the humidity has to be uniformly distributed from 0 to 99, adding remarkable noise. To validate this, we also examine their performance with the homogeneous data, and the results show that the two algorithms perform almost the same.

In comparison, C4.5 algorithm achieves better accuracy than the ID3 algorithm, this is because C4.5 can deal with numerical attribute, whereas in ID3, the numerical attribute is considered as category attribute, thus reducing the accuracy in order to avoid the decision tree becoming too big.

4.5. Impact of training data distribution

To further understand the impact of the distribution of the data set, we perform comparative experiments with both the heterogeneous data and the homogeneous data. Figs. 8 and 9 plot their respective accuracy results with different noises.

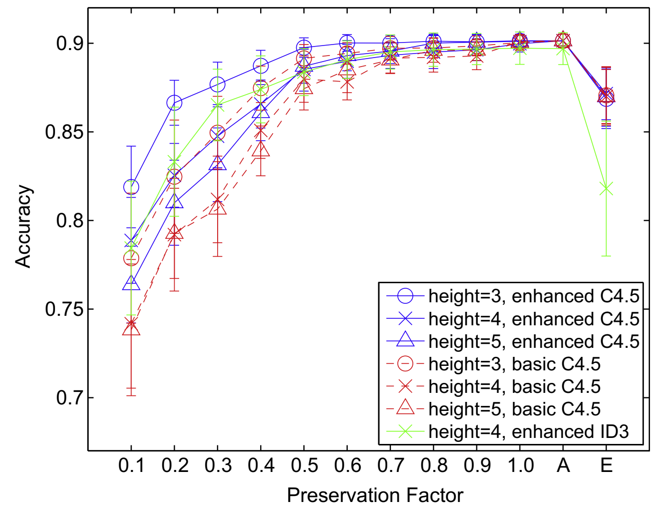

Fig. 7. Comparison of enhanced and basic C4.5 algorithms for heterogeneous data with $\varepsilon = 10\%$.

We find that for the heterogeneous data, when the preservation factor is small, the accuracy is low. On the other hand, for the homogeneous data, the factor does not affect the accuracy. For data with small noise, when the preservation factor is greater than 0.5, the two achieve similar accuracy. If the noise is larger, the preservation factor has to go beyond 0.8 to achieve the similar accuracy. This is because in homogeneous data distribution, all the nodes have similar training data, thus the classifiers built by the nodes are almost the same, which leads to the high similarity between the generated pseudo data and the local training data. Therefore, the accuracy is independent on the preservation factor for homogeneous data.

In the ensemble method, the accuracy for heterogeneous data is much lower than that of homogeneous data and our hierarchical approach. This has been explained in the first experiment. For the homogeneous data, the ensemble method has the same accuracy as our approach.

4.6. Comparison of energy consumption

Finally, we investigate the energy consumption, which is one of the most important concerns in wireless sensor networks. We

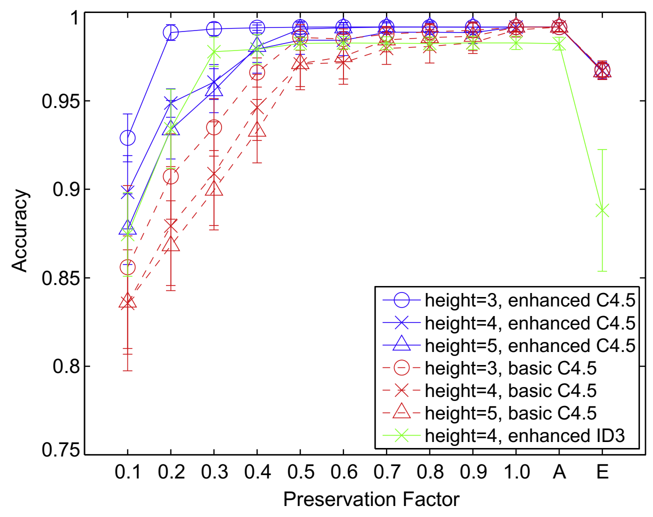

Fig. 6. Comparison of enhanced and basic C4.5 algorithms for heterogeneous data with $\varepsilon = 1\%$.

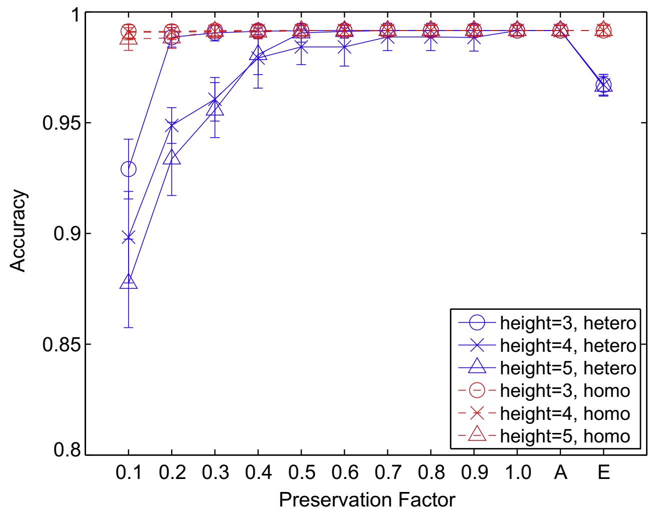

Fig. 8. Comparison of heterogeneous and homogeneous data distribution with $\varepsilon = 1\%$.

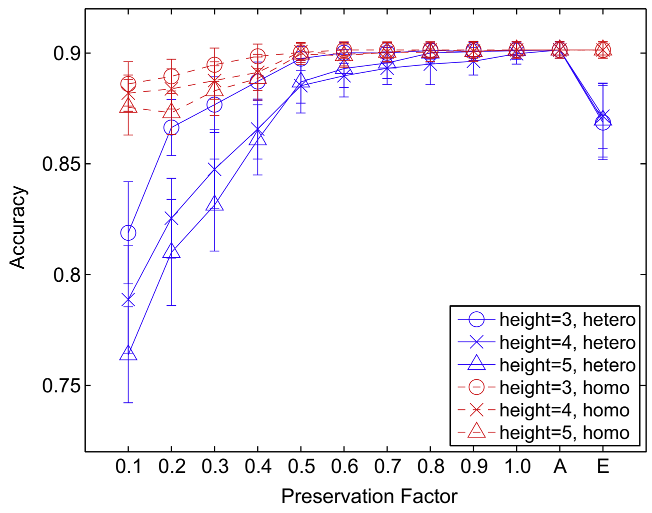

Fig. 9. Comparison of heterogeneous and homogeneous data distribution with $\varepsilon = 10\%$.

compare our hierarchical classification with the ensemble method that transmits all the classifiers to the base station. The energy consumption consists of the computation energy consumption and the transmission energy consumption, which is *significantly* larger than the prior one [8], thus we neglect the computation energy consumption. The transmission energy consumption depends on the size of the transmitted data and the distance between the two nodes. Generally, it is proportional to the data size and square of the distance [8]. We normalize the result and plot in Fig. 10.

From the figure, we find that the energy consumption of our approach is much lower than the ensemble method in all the situations when the height is greater than 1. On average, our method saves nearly 70% of the energy spent in the ensemble method when height is greater than 1. The greater the height is, the more energy we save, simply because a classifier is only forwarded to the parent in our solution, while it is forwarded all the way to the base station in the ensemble method.

We find that the data distribution does not significantly affect the energy consumption and there is no consistent observation. This is different from our previous work that based on ID3 algorithm, in which the data from a node are likely to exist in one branch for heterogeneous data. Because the decision tree is pruned after creation in C4.5, for homogeneous data, the leaf nodes are likely to be merged if the sibling leaf nodes' class labels are the same.

According to the figures, the noise does not affect the energy consumption much. We believe that it is because the noise does not change the transmission distance, and it also does not noticeably change the size of the decision trees.

As we modify the basic C4.5 algorithm, the constructed decision tree is thus larger in our enhanced C4.5 algorithm. We also examine the size of the decision tree built in the two algorithms, and find that the size in enhanced C4.5 is 18.7% larger than that in the basic C4.5 algorithm. Compared with our previous work, in which the enhanced ID3 algorithm is also nearly 45% larger than the basic ID3 algorithm, we can see the improvement of enhanced C4.5 over enhanced ID3. Moreover, considering that the basic C4.5 algorithm cannot even work due to its low accuracy in our application, this small increased size is reasonably acceptable.

5. Related work

There are many classic methods for centralized classification, such as bagging [17] and boosting [18]. Random forest [19] is another advanced tool, which combines tree predictors, such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Random inputs and random features produce good results in the classification.

For distributed data classification using the ensemble method, several techniques have been proposed [20,21]. Distributed classification in peer-to-peer networks is also studied [16]. The authors proposed an ensemble approach, in which each peer builds its local classifiers on the local data and then combines all the classifiers by plurality voting. These distributed solutions however are not customized for sensor networks. In particular, they do not consider the limit of memory sizes, nor the energy consumption, which are two critical concerns with battery-powered sensor nodes. Directly applying these distributed classifications into our application scenario will thus result in poor performance, as demonstrated through our simulations.

There have been numerous works on data gathering in wireless sensor networks. For example, LEACH [8] and PEGASIS [22] attempt to make the data collection task energy efficient. There are also a few related works about classification in this context [23–

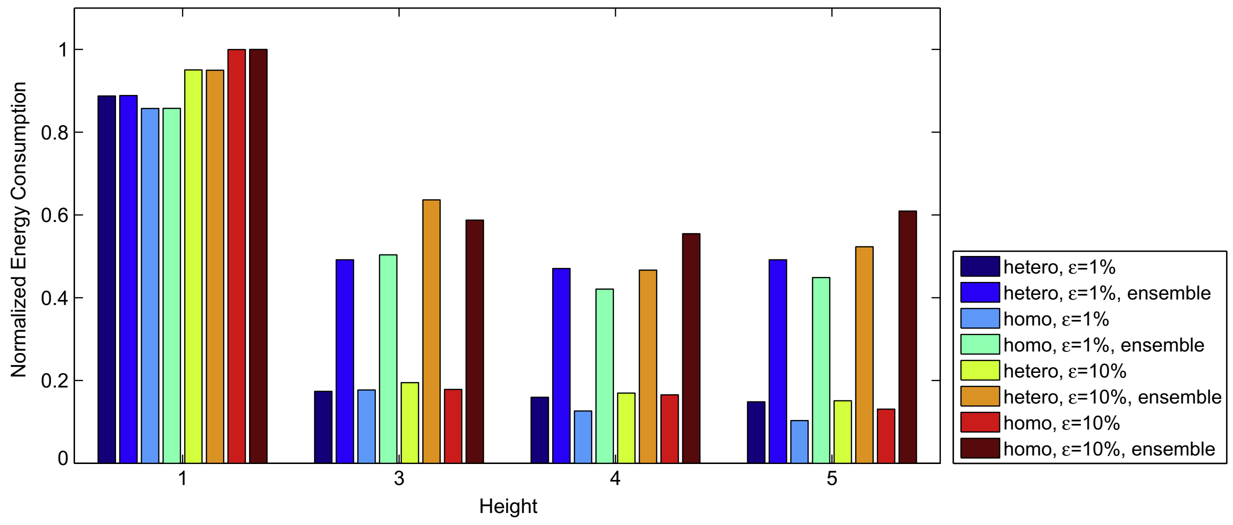

Fig. 10. Comparison of energy consumption.

25], but they have focused on detecting or tracking objects, while not giving detailed design of classifiers. To the best of our knowledge, our work is the first addressing energy-efficient distributed classification for wireless sensor networks, particularly with heterogeneous data distribution across the sensor nodes.

This work extends our previous work [14] by presenting the detailed spanning tree constructing approach, and adopting an advanced C4.5 algorithm for the classification basic. Therefore, numerical attribute such as temperature and humidity can be better accommodated by our enhanced C4.5 algorithm, achieving better classification accuracy; the pruning procedure in C4.5 also reduces the decision tree size, thus saving more energy.

6. Conclusion

In this paper, we have proposed a novel decision-tree-based hierarchical distributed classification approach in wireless sensor networks. In particular, we consider a practical scenario in which the data distribution is heterogeneous, which is seldom studied before. In our solution, the sensor nodes are organized in a spanning tree, and local classifiers are built by individual sensor nodes and merged along the routing path. The classifiers are iteratively enhanced by combining strategically generated pseudo data with new local data, eventually converging to a global classifier for the whole network. We demonstrate that our approach maintains high classification accuracy with very low storage and communication overhead.

With the framework, an intelligent wildlife monitor can be developed. It will not only sense and track the animal, but also perform classification that mines the environment suitability. However, there could be many possible enhancements. We are particularly interested in designing a more effective pseudo data generating algorithm that makes the generated data as close to the original as possible with limited side information. We are also interested in evaluating the performance with real sensor testbeds.

Acknowledgment

This research is supported by a Canadian NSERC Discovery Grant, an NSERC Discovery Accelerator Grant and an NSERC Strategic Project Grant.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Communications Magazine 40 (8) (2002) 102–114.

[2] J. Yick, B. Mukherjee, D. Ghosal, Wireless sensor network survey, Computer Networks 52 (12) (2008) 2292–2330.

[3] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson, Wireless sensor networks for habitat monitoring, in: Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA), 2002, pp. 88–97.

[4] X. Liu, Q. Huang, Y. Zhang, Combs, Needles, haystacks: balancing push and pull for discovery in large-scale sensor networks, in: Proceedings of International Conference on Embedded Networked Sensor Systems (SenSys), 2004, pp. 122–133.

[5] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, J. Lees, Deploying a wireless sensor network on an active volcano, IEEE Internet Computing 10 (2) (2006) 18–25.

[6] E. Cayirci, T. Coplu, SENDROM: sensor networks for disaster relief operations management, Wireless Networks 13 (3) (2007) 409–423.

[7] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, Next century challenges: scalable coordination in sensor networks, in: Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), 1999, pp. 263–270.

[8] W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan, An application-specific protocol architecture for wireless microsensor networks, IEEE Transactions on Wireless Communications 1 (4) (2002) 660–670.

[9] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Addison Wesley Publishers, 2005.

[10] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufman Publishers, 2006.

[11] J.R. Quinlan, Induction of Decision Trees, in: Machine Learning, Kluwer Academic Publishers., 1986, pp. 81–106 (Chapter 1).

[12] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufman Publishers, 1993.

[13] F. Bouhafs, M. Merabti, H. Mokhtar, A node recovery scheme for data dissemination in wireless sensor networks, in: Proceedings of IEEE International Conference on Communications (ICC), 2007, pp. 3876–3881.

[14] X. Cheng, J. Xu, J. Pei, J. Liu, Hierarchical distributed data classification in wireless sensor networks, in: Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS), 2009.

[15] T.G. Dietterich, Ensemble methods in machine learning, in: Proceedings of the First International Workshop on Multiple Classifier Systems (MCS), 2000, pp. 1–15.

[16] P. Luo, H. Xiong, K. Lu, Z. Shi, Distributed classification in peer-to-peer networks, in: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), 2007, pp. 968–976.

[17] L. Breiman, Bagging predictors, Machine Learning 24 (2) (2002) 123–140.

[18] Y. Freund, Boosting a weak learning algorithm by majority, in: Proceedings of Workshop on Computational Learning Theory, 1990, pp. 202–216.

[19] L. Breiman, Random forests, Machine Learning 45 (1) (2001) 5–32.

[20] A. Lazarevic, Z. Obradovic, The distributed boosting algorithm, in: Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), 2001, pp. 311–316.

[21] N.V. Chawla, L.O. Hall, K.W. Bowyer, W.P. Kegelmeyer, Learning ensembles from bites: a scalable and accurate approach, Machine Learning Research 5 (2004) 421–451.

[22] S. Lindsey, C. Raghavendra, K.M. Sivalingam, Data gathering algorithms in sensor networks using energy metrics, Transactions on Parallel and Distributed Systems 13 (9) (2002) 924–935.

[23] R.R. Brooks, P. Ramanathan, A.M. Sayeed, Distributed target classification and tracking in sensor networks, Proceedings of the IEEE 91 (8) (2003) 1163–1171.

[24] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, M. Miyashita, A line in the sand: a wireless sensor network for target detection, classification, and tracking, Computer Network 46 (5) (2004) 605–634.

[25] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J.A. Stankovic, T. Abdelzaher, B.H. Krogh, Lightweight detection and classification for wireless sensor networks in realistic environments, in: Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys), 2005, pp. 205–217.