Embedded Machine Learning-Based Data Reduction in Application-Specific Constrained IoT Networks

by

adarshpal.singh , Sachin Chaudhari

in

The 35th ACM/SIGAPP Symposium On Applied Computing

Brno, Czech Republic

Report No: IIIT/TR/2020/-1

Centre for Communications
International Institute of Information Technology
Hyderabad - 500 032, INDIA
March 2020

Embedded Machine Learning-Based Data Reduction in Application-Specific Constrained IoT Networks

Adarsh Pal Singh
International Institute of Information Technology (IIIT)
Hyderabad, India
adarshpal.singh@research.iiit.ac.in

Sachin Chaudhari
International Institute of Information Technology (IIIT)
Hyderabad, India
sachin.chaudhari@iiit.ac.in

ABSTRACT

Reducing the amount of wireless data transmissions in constrained battery-powered sensor nodes is an effective way of prolonging their lifetime. In this paper, we present a machine learning-based data transmission reduction scheme for application-specific IoT networks. Though many error thresholding-based data prediction schemes have been explored in the past, this is the first work to incorporate machine learning in constrained sensor nodes to reduce data transmissions. We also provide a generic overview and comparison of five traditional supervised machine learning algorithms in the context of offloading trained models to memory and computationally constrained microcontrollers. The proposed data reduction scheme is validated on an occupancy estimation testbed deployed in our lab. Experimental results demonstrate 99.91% overall reduction in data transmissions while imparting similar performance and 18 to 82 times lesser transmissions than Shewhart change detection algorithm.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Computer systems organization → Sensor networks**; *Embedded systems*;

KEYWORDS

Internet of things, wireless sensor networks, data transmission reduction, machine learning, distributed detection, embedded systems

ACM Reference Format:
Adarsh Pal Singh and Sachin Chaudhari. 2020. Embedded Machine Learning-Based Data Reduction in Application-Specific Constrained IoT Networks. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341105.3373967

1 INTRODUCTION

Numerous internet of things (IoT) applications require wireless sensor networks (WSNs) to be deployed in constrained environments devoid of regular wired power supply. Naturally, such applications have boosted the usage of battery-powered and energy harvesting-based sensor nodes. These nodes typically comprise of a memory and computationally constrained low-power microcontroller, sensor(s) and a radio frequency (RF) transceiver. The radio transmission is known to be the biggest consumer of energy in WSNs [12]. Consequently, increasing the lifetime of these constrained wireless sensor nodes by decreasing the amount of data transmissions without compromising much on the data quality is an important area of research.

In a typical star WSN, the sensor nodes transmit data to the sink (or the edge) periodically while following a *sleep* → *wake up* → *sense* → *transmit* → *sleep* schedule. Several data compression techniques have been explored for data reduction in such networks [17]. However, even though these algorithms shorten the data packets, the transmission periodicity remains the same. Since the sensors themselves are bounded by an accuracy threshold, some amount of error can generally be tolerated by the user. Hence, a majority of data reduction schemes for WSNs revolve around data prediction models that run on both the sensor nodes and the sink and initiate transmission only when the predicted value exceeds the current sensed value by a predefined error threshold [5]. In [9], the authors proposed a dual Kalman filter (DKF) for WSNs. The filter weights require prior statistical information about the phenomenon under measurement in the training phase and then the trained model is offloaded to both the sensor node and the sink to perform predictions in real time. A simpler and more popular adaptive filter approach that doesn't require prior information about the data was proposed by authors in [3]. Among different adaptive filter algorithms, least mean square (LMS) is widely used since it exhibits a lower computational cost and more flexibility [19]. Time series-based prediction algorithms such as moving average (MA), exponential weighted moving average (EWMA), autoregressive integrated moving average (ARIMA) and the simplest naive algorithm (also known as Shewhart change detection or simply Shewhart [11]) have also been explored for data reduction in WSNs [1]. Interestingly, the authors in [1] experimentally showed Shewhart to be more optimal than MA, EWMA and even ARIMA both in the number of transmissions and the root mean squared error (RMSE) metric for temperature data. Shewhart has also been shown to outperform LMS in [2, 16]. Hence, we have used Shewhart as a benchmark for our proposed algorithm.

The data prediction algorithms explored in the WSN literature so far are all thresholding-based schemes that do not take data importance into account when deciding the initiation of transmission, not even for application-specific IoT networks. The authors in [7] recently pioneered the work on machine learning (ML)-based data importance calculation to reduce the number of transmissions for an IoT vehicle mobility use case. However, their work is focused on decreasing the volume of data transmitted by resource-abundant IoT devices such as smart cars and smartphones and not on increasing the lifetime of constrained micrcontroller-based sensor nodes. Embedded ML has been explored in the past [6, 8, 10] but not in the context of data transmission reduction. Also, there is a dearth of a generic study that compares the run times and memory constraints of different supervised ML algorithms in the context of microcontrollers.

In this paper, we present an overview and comparison of five traditional supervised ML algorithms namely linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), Gaussian naive Bayes (GNB), support vector machine (SVM) and decision tree (DT) in the context of microcontrollers and present an ML-based data reduction framework for application-specific constrained IoT networks. The proposed framework is validated on an occupancy estimation testbed and compared to the Shewhart-based data reduction scheme in terms of accuracy, F1 score and the number of transmissions.

The rest of the paper is structured as follows. Section 2 describes our proposed data reduction scheme and presents a study on ML algorithms in the context of constrained embedded microcontrollers. Section 3 describes our experimental testbed and the logistics of the experiments performed to validate our proposed data reduction scheme. Section 4 juxtaposes the results of different experiments to provide reasoning and logical comparisons. The paper is concluded in Section 5.

2 METHODOLOGY

2.1 Proposed Approach

Supervised ML algorithms require a substantial amount annotated data and vast computing resources in the training phase, far greater than what a typical microcontroller can provide. However, once trained, many of these algorithms boil down to simple parameters that require standard arithmetic and logical operations for inference (testing). This can easily be run on microcontrollers in real time. But unlike signal processing and time series-based data reduction algorithms that can run in any scenario, this method requires a specific application out of the sensor network since annotated training data is needed to train the supervised ML models. Once trained, the model parameters can be offloaded to the sensor node(s) which can then run the inference in real time.

In a typical constrained WSN, multiple battery-powered or energy harvesting-based sensor nodes are scattered in a region with a sink/edge device in range to gather data and take care of automation. The sensor nodes usually practice a periodic sleep-wake cycling pattern to conserve energy. Without any data reduction scheme, the array comprising of sensor data $x[n]$, where n is the number of quantities sensed by the node, would be transmitted to the edge each time the sensor node wakes up. In this case, the data analytics part is taken care of by the edge or the cloud. In case where a data reduction scheme is employed, the decision of whether to initiate transmission or not is typically governed by a mathematical model involving the current sensed array $x[n]$, the previous sensed array and/or the previous transmitted array. In our case, since a trained ML model is present at the sensor node itself,

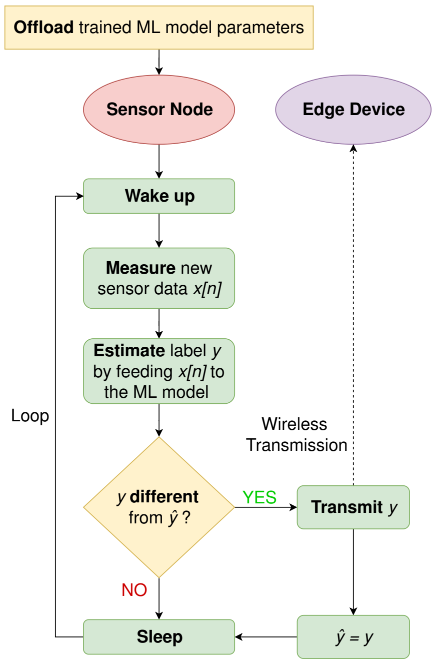
Offload trained ML model parameters
Sensor Node
Edge Device
Wake up
Measure new sensor data *x[n]*
Estimate label *y* by feeding *x[n]* to the ML model
Loop
Wireless Transmission
y **different** from *ŷ* ?
YES
Transmit *y*
NO
Sleep
ŷ = *y*

Figure 1: Algorithmic flowchart for data transmission reduction using machine learning on wireless sensor nodes.

$x[n]$ is tested against the model to find the inference label y here itself. Let the previously transmitted label be $\hat{y}$. The inference y is only transmitted in case y differs from $\hat{y}$. This way, the edge device always has the current scenario of the application at hand with the least amount of transmissions from the sensor node. Figure 1 shows the flow of the proposed algorithm.

2.2 Machine Learning Algorithms

We present an overview of five traditional ML algorithms with specific insights into offloading the model parameters and running the inference on a microcontroller. All of these algorithms are tested for time and space complexity on Atmel ATmega328P, an 8-bit microcontroller. For the scope of this paper, our focus will only be on binary classification.

2.2.1 Linear Discriminant Analysis (LDA). LDA is a simple yet popular linear classifier that assumes the class conditional distribution of the data, $P(X = x|Y = k)$, where X represents the training set and Y denotes some class k out of the set of classes $\{1, \ldots, K\}$, as a multivariate Gaussian.

$$f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)} \quad (1)$$

The d in the above equation signifies the dimension or the number of features that the data has, μ_k is the mean vector of the data under class k and Σ is the covariance matrix which is common for all classes. For a new sensed data vector x_s, predictions are made using Bayes' rule-

$$P(Y=k|X=x_s) = \frac{f_k(x_s)P(y=k)}{\sum_{i=1}^{K} f_i(x_s)P(y=i)} \tag{2}$$

In the training phase of LDA, class means μ_k, covariance matrix Σ and class priors $P(y=k)$ are learnt from the training data. This phase is carried out either at the edge or the cloud. In scikit-learn's [14] implementation of LDA, post training, one can get these values from the *means_*, *covariance_* and *priors_* attributes respectively. For inference, the class k which maximizes the above equation (note that the denominator is same for all the classes) is chosen as the predicted class. So, for binary classification ($Y \in \{0,1\}$), inference is made using the rule-

$$P(Y=1|X=x_s) \underset{0}{\overset{1}{\gtrless}} P(Y=0|X=x_s) \tag{3}$$

Putting Equations (1) and (2) in (3) and simplifying, binary class inference in LDA boils down to a simple linear equation-

$$x_s^T.w \underset{0}{\overset{1}{\gtrless}} c \tag{4}$$

where $c = ln\frac{P(y=0)}{P(y=1)} - \frac{1}{2}(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1^T\Sigma^{-1}\mu_1)$ is a 4-byte floating-point constant and $w = \Sigma^{-1}[\mu_1 - \mu_0]$ is a constant $d \times 1$ floating-point vector. Both of these constants can be calculated from the aforementioned formulations post the training phase and then offloaded to the microcontroller. Time complexity-wise, d floating-point multiplications, $d-1$ floating-point additions and only one comparison is required.

2.2.2 *Quadratic Discriminant Analysis (QDA).* As the name suggests, QDA is a quadratic decision boundary classifier and has the same formulation as LDA except for the fact that the covariance matrix Σ is calculated separately for each class. Therefore, the class conditional distribution of the data in this case is formulated as-

$$f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)} \tag{5}$$

As before, for any new sensed data vector x_s, predictions are made using Bayes' rule (Equation (2)) and the inference for binary classification is made using Equation (3). Putting Equations (2) and (5) in (3) and simplifying, QDA inference boils down to-

$$x_s^T\Sigma_{diff}^{-1}x_s + x_s^T w \underset{0}{\overset{1}{\gtrless}} c \tag{6}$$

where, $c = 2ln\frac{P(y=0)|\Sigma_1|^{\frac{1}{2}}}{P(y=1)|\Sigma_0|^{\frac{1}{2}}} + \mu_1^T\Sigma_1^{-1}\mu_1 - \mu_0^T\Sigma_0^{-1}\mu_0$ is a 4-byte floating-point constant, $\Sigma_{diff}^{-1} = \Sigma_0^{-1} - \Sigma_1^{-1}$ is a $d \times d$ constant floating-point matrix and $w = 2\Sigma_1^{-1}\mu_1 - 2\Sigma_0^{-1}\mu_0$ is a $d \times 1$ constant floating-point vector. These constants can be calculated post training and then offloaded to the microcontroller. Time complexity-wise, (d^2+2d) multiplications, (d^2+d-1) additions and one comparison is required. Clearly, this is much more memory and computationally expensive than LDA but easily doable for low dimensional data. As per our experiment which is described in detail in the later part of this section, it is not possible to go beyond 17 dimensions with QDA on Atmega328P since it runs out of memory.

2.2.3 *Gaussian Naive Bayes (GNB).* Naive Bayes', as the name suggests, is based on the Bayes' theorem coupled with the "naive" assumption that all the d features of the dataset given the class label are mutually independent. Let x_s be a d-dimensional data vector. Then, $x_s = (x_1, x_2, ..., x_d)$. Using Bayes' theorem,

$$P(Y=y|X=x_s=(x_1,...,x_d)) = \frac{P(Y=y)P(x_1,...,x_d|y)}{P(x_1,...,x_d)} \tag{7}$$

The denominator in the above equation is same for all the classes. Using the conditional independence of the features, we can formulate a classifier as follows-

$$\hat{Y} = \arg\max_y P(Y=y)\prod_{i=1}^{d} P(x_i|y) \tag{8}$$

Under GNB, the distribution of the features given a class label is assumed to be Gaussian. Therefore,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_{y,i}^2}} e^{-\frac{(x_i-\mu_{y,i})^2}{2\sigma_{y,i}^2}} \tag{9}$$

where $\sigma_{y,i}^2$ and $\mu_{y,i}$ denote the variance and mean of feature i under class y respectively. In scikit-learn's implementation of GNB, the feature variances, means and the class priors $P(Y=y)$ can be taken from *sigma_*, *theta_* and *class_prior_* attributes respectively. For binary labels, we can rewrite Equation (8) as-

$$P(y=1)\prod_{i=1}^{d} P(x_i|y=1) \underset{0}{\overset{1}{\gtrless}} P(y=0)\prod_{j=1}^{d} P(x_j|y=0) \tag{10}$$

Putting Equation (9) in (10) and simplifying, we end up with the following inference rule-

$$\sum_{j=1}^{d} \rho_{0,j}^2(x_j-\mu_{0,j})^2 - \sum_{i=1}^{d} \rho_{1,i}^2(x_i-\mu_{1,i})^2 \underset{0}{\overset{1}{\gtrless}} c \tag{11}$$

where $c = 2ln\frac{P(y=0)\prod_{j=1}^{d}\frac{1}{\sqrt{\sigma_{0,j}^2}}}{P(y=1)\prod_{i=1}^{d}\frac{1}{\sqrt{\sigma_{1,i}^2}}}$ is a 4-byte floating-point constant, $\rho_0^2 = \frac{1}{\sigma_0^2}$ and $\rho_1^2 = \frac{1}{\sigma_1^2}$ are d-length floating-point arrays. The inverse of variances are taken to convert the division operations to multiplications since the latter is much more computationally efficient in microcontrollers. So post training, we need to offload four d-length floating-point arrays: μ_0, μ_1, ρ_0^2 and ρ_1^2 and a floating-point constant c. Time complexity-wise, $4d$ multiplications, $4d-1$ additions/subtractions and one comparison is required.

2.2.4 *Support Vector Machine (SVM).* SVM is one of the most popular supervised machine learning classifiers. Unlike LDA, QDA and GNB, SVM does not make any assumptions about the distribution of the data. Rather, it simply attempts to fit an optimal hyperplane that separates one class from the other [4]. Since our focus is only on inference, we shall skip the optimization formulation that is used during the training. For any new sensed data vector x_s, we just need to check in which half of the separating hyperplane this vector lies in-

$$w^T\bar{x}_s \underset{0}{\overset{1}{\gtrless}} c \tag{12}$$

Here, w is the $d \times 1$ floating-point weight vector (coefficients of the hyperplane equation), c is the negative of the intercept of the hyperplane and $\bar{x}_s$ is the standard-scaled transformed vector of x_s which is calculated as-

$$\bar{x}_s[i] = \frac{x_s[i] - u[i]}{s[i]}, \quad \forall i \in \{1, 2, ..., d\} \tag{13}$$

where u and s are the d-length floating-point arrays having means and standard deviations of the training samples respectively. As it was done in the GNB case, the more complex division operation is converted to multiplication by taking the inverse of s so that the standard scaling transformation can be done by-

$$\bar{x}_s[i] = p[i](x_s[i] - u[i]), \quad \forall i \in \{1, 2, ..., d\} \tag{14}$$

This scaling of data is necessary since the SVM algorithm is scale-variant. Hence, apart from w and c, scaling parameters u and p also need to be offloaded. During the training phase, the first step is training data standardization which is usually done using scikit-learn's *StandardScaler* function. Once the training data is fitted and transformed, attributes *mean_* and *scale_* will give u and s respectively (p can be calculated from s by taking the inverse of each element). After linear SVM is trained on this scaled data, one can get w from the *coef_* attribute and c from the negative of *intercept_* attribute. Note that the inference operation of kernelized non-linear SVM is too complex for simple microcontrollers and hence we are only sticking with linear SVM. Time complexity-wise, $2d$ multiplications, $2d - 1$ additions/subtractions and a single comparison is required for inference.

2.2.5 Decision Tree (DT). Decision Tree, in simple terms, learns optimum decision rules from the training set and discriminates between different class labels based on these rules. In a nutshell, a trained DT model is an *if-else* ladder in the shape of a tree with branches feeding to test nodes (a logical test is performed on the value of a particular feature) and the leaves corresponding to one of the class labels. Unlike the algorithms listed above, only the logical comparison operation is needed for inference here. Once trained, each root-to-leaf path of the decision tree can be represented as-

if condition$_1$ *and* condition$_2$ ... *and* condition$_k$ *then* label

Here, each condition is a node in the path from root to one of the leaves (assuming k nodes including the root and excluding the leaf). The label is the corresponding leaf node. This conversion has to be done for all the unique root-to-leaf paths in the trained DT and then offloaded to the microcontroller. This can be done manually by printing the trained DT model. Note that the number of root-to-leaf paths (or the number of *if* statements) would be equal to the number of leaves. Therefore, the inference memory and time complexity is dependent on the number of leaf nodes (N_{leaf}) which is same as the number of *if* statements and the average depth (D_{avg}) of the tree which is nothing but the average number of comparisons in a single *if* statement. Both of these parameters can be controlled during the training phase. In the scikit-learn implementation of DT, *max_depth* and *max_leaf_nodes* parameters can be specified to control the growth of the tree.

Table 1 summarizes the inference complexities of the ML algorithms discussed so far. Figure 2 shows the execution times of LDA, QDA, SVM and GNB as a function of the dimensionality of the data. These experiments were performed on ATmega328P and the execution times were calculated by leveraging the internal timer which has a resolution of 4 μs. DT has been omitted from the plot since it is not directly dependent on the data dimension. However, just for reference, for $d = 3$, our DT model took 48 μs when trained with *min_samples_split* = 50. Note that the execution time of the QDA inference doesn't exist for $d > 17$ since Atmega328P runs out of memory beyond that. LDA and SVM, being simple linear classifiers, take the least amount of time for inference. LDA is a tad bit lighter as well as faster than SVM since the data points in case of LDA do not require scaling.

Table 1: Complexities of different ML algorithms

Algorithm	Multiplications	Additions	Bytes Offloaded
LDA	d	$d-1$	$4d+4$
QDA	d^2+2d	d^2+d-1	$4d^2+4d+4$
GNB	$4d$	$4d-1$	$16d+4$
SVM	$2d$	$2d-1$	$12d+4$
DT	Nil	Nil	$4D_{avg}N_{leaf}$

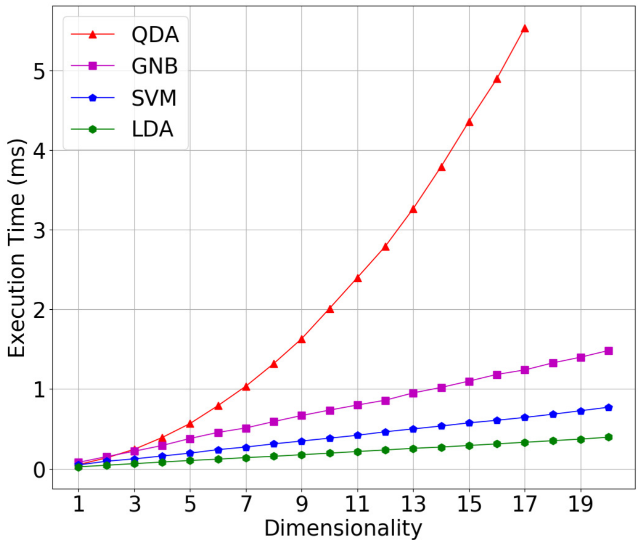

Figure 2: Execution times of 4 ML algorithms with respect to the dimensionality of the data as measured on Atmel ATmega328P.

3 EXEMPLAR USE CASE

The data transmission reduction framework described in the earlier sections is generic and can be adapted to many IoT use cases. For a proof of concept, we take up one of the most explored applications in the IoT domain: estimating the number of humans in a room using non-intrusive sensors. The experimental testbed and the logistics of the experiments on occupancy estimation and data reduction is described in this section.

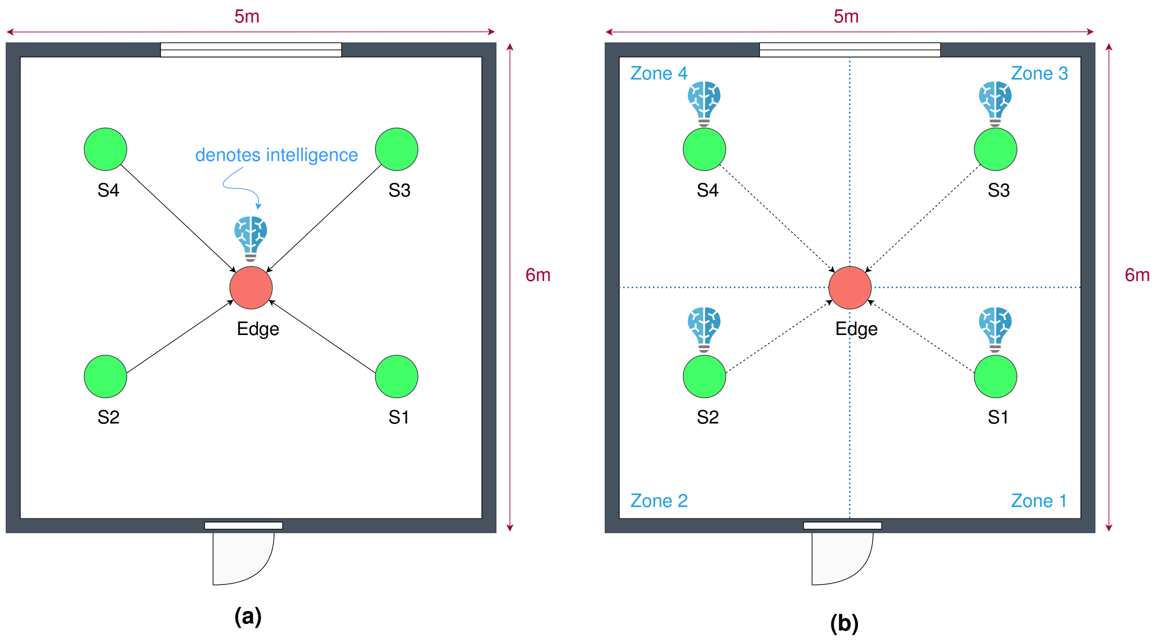

Figure 3: Occupancy estimation testbed depicting the sensor network. 3(a) is a reference for experiments 1 and 2 wherein the intelligence resides at the edge. 3(b) is a reference for experiment 3 wherein the sensor nodes themselves transmit data intelligently.

3.1 Sensor Network Testbed

Figure 3(a) shows our experimental testbed comprising of four sensor nodes (S1, S2, S3 and S4) connected wirelessly to an edge device in a small lab. This is essentially a star topology-based WSN. Each sensor node comprises of-

(1) An Atmel ATmega328P 8-bit microcontroller
(2) Three sensors- temperature (DS18B20), humidity (DHT22) and light (BH1750)
(3) XBee S2C (Zigbee wireless transciever)

The edge device is a Raspberry Pi 3 Model B with an XBee S2C for receiving data. This occupancy estimation setup is inspired from [13, 18] which demonstrate the need of having multiple multivariate sensor nodes.

Since we are using an ML-based paradigm, training data is required. So the first phase after the deployment was the data collection phase. Each sensor node was programmed to periodically send sensor data to the edge every 30s in a *sleep* $\longrightarrow$ *wake up* $\longrightarrow$ *sense* $\longrightarrow$ *transmit* $\longrightarrow$ *sleep* fashion. No data reduction scheme was employed in this phase. The edge simply logged the data it received from each sensor node into its own separate .csv file. The ground truth of the number of people in the room as well as in the specific sensor zones (Figure 3(b)) was recorded manually. The reason as to why the latter was necessary will be clear in the later part of this paper. The data collection phase lasted for around 4 days and after cleaning the raw data of missing and erroneous values, we were left with a little more than 10,000 points for each sensor node. The ground truth of the number of people ranged from 0-3 with at most one person in each zone. Note that the zone 4 was kept empty for the entirety of the data collection phase as we wanted to

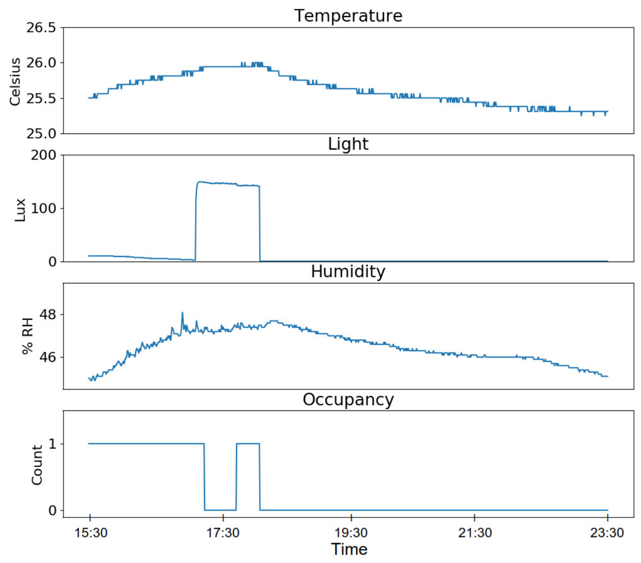

Figure 4: An 8-hour time series plot of S1 sensor data juxtaposed with the zone 1 occupancy count.

observe the correlations between the occupants in different zones with the data from an undisturbed sensor node. To get a sense of how different sensor features correlate with occupancy, an 8-hour time series plot of S1 sensor data juxtaposed with the occupancy count is shown in Figure 4.

Table 2: Accuracy and F1 score of different ML algorithms under CD, SRD and DD experiments.

Experiment	Metric	SVM (RBF)	SVM (Lin)	LDA	QDA	GNB	DT	RF
CD	A	**0.977**	0.964	0.972	0.940	0.892	0.960	0.976
	F1	**0.939**	0.914	0.926	0.886	0.738	0.886	0.924
SRD1	A	**0.972**	0.962	0.972	0.852	0.872	0.921	0.969
	F1	**0.928**	0.909	0.926	0.815	0.717	0.825	0.907
SRD2	A	**0.972**	0.962	0.972	0.889	0.869	0.943	0.967
	F1	**0.931**	0.909	0.926	0.838	0.717	0.865	0.901
SRD3	A	0.970	0.960	**0.972**	0.786	0.866	0.905	0.964
	F1	0.922	0.903	**0.924**	0.781	0.721	0.807	0.891
SRD4	A	**0.969**	0.960	0.967	0.804	0.872	0.925	0.964
	F1	**0.922**	0.901	0.901	0.777	0.728	0.830	0.895
DD	A	N/A	**0.968**	0.965	0.933	0.915	0.890	N/A
	F1	N/A	**0.916**	0.907	0.812	0.759	0.743	N/A

3.2 Logistics of the Experiments

With the collected dataset, three different experiments were simulated to validate our idea.

3.2.1 Exp 1: Occupancy Estimation with Complete Dataset (CD). This experiment simulates the original experiments in [13, 18] with the edge doing the real-time occupancy predictions using machine learning models trained on the complete dataset with no data reduction scheme running on the sensor nodes. The dataset in this case was made by combining all the four sensor node files into one and the room occupancy count was used as the classifying label. In general, a single ML model pertaining to the complete dataset is more powerful than separate models for each sensor node as the latter is restricted to heterogeneous fusion of sensory data in a single location alone while the former also exploits patterns via homogeneous and heterogeneous fusion across spatially scattered sensor nodes. For example, in Figure 3(b), if there is an occupant in zone 1 then the sensor nodes in zones 2, 3 and 4 have been observed to be affected by it. The extent of this affect is, of course, dependent on the sensor type and the distance between the occupant and the sensor node. Since ML is running on the edge, there is no restriction on the complexity of the model. Therefore, apart from the 5 ML models listed in the previous section, SVM with RBF kernel and Random Forest (RF) are also used. Accuracy and F1 score metrics were evaluated by doing a 10-fold stratified cross validation on the complete dataset. Shuffling of the data was not done since the dataset is of time series nature.

3.2.2 Exp 2: Occupancy Estimation with Shewhart-Reduced Dataset (SRD). This experiment simulates Shewhart-based data reduction scheme running on the sensor nodes. In Shewhart, an error threshold is defined on each of the sensors and if the current value of a sensor exceeds or goes below the previously transmitted value by that error threshold, the data is sent. Therefore, if the edge doesn't receive any data from the sensor node in a particular interval, it assumes the previously transmitted data to be true for that interval as well. Clearly, the data analytics part in this case is also handled by the edge and thus a full model can be exploited as in the previous experiment. For our setup, we have 3 error thresholds, one for each sensor (e_t for temperature, e_h for humidity and e_l for light). The error thresholds are usually derived from the sensor resolution (10 times the resolution in general), sensor accuracy and mean of the data collected (2-5% in general). Based on this, we arrive at four different experiments for the SRD case-

(1) SRD1: $e_t = 0.5$, $e_h = 1.0$ and $e_l = 1$.
(2) SRD2: $e_t = 0.5$, $e_h = 2.0$ and $e_l = 1$.
(3) SRD3: $e_t = 0.5$, $e_h = 1.0$ and $e_l = 10$.
(4) SRD4: $e_t = 0.5$, $e_h = 2.0$ and $e_l = 10$.

The accuracy and F1 metrics for this case have again been calculated using a stratified 10-fold cross validation approach on the complete dataset. In each iteration of the 10-fold loop, the testing set was simulated to run Shewhart change detection separately for each sensor node and combined into a single vector based on the outcome of the test for each of the testing data points.

3.2.3 Exp 3: Occupancy Estimation with ML-Reduced Distributed Detection (DD). This experiment simulates our proposed approach wherein each sensor node runs its own ML model and only sends the inference when it differs from the previously transmitted inference. Hence, the data analytics part is taken care of by the sensor nodes themselves. Since each sensor node works independently without sharing data with other nodes, this is essentially a distributed detection problem. The ML model for each sensor node is trained on the data from that sensor node and the corresponding zonal occupancy count (Figure 3(b)). The edge, in this case, has the occupancy count of each zone and simply sums it up for all the zones to find the total room occupancy. To calculate the accuracy and F1 score for this case, stratified 10-cross validation approach was applied to each of the sensor node models (separately trained) and the predictions were summed up to correspond to the room occupancy.

4 RESULTS

The ML experiments described in the previous sections were performed with scikit-learn [14]. Macro F1 score is used instead of micro F1 score since the dataset is skewed with more points corresponding to the zero occupancy case than the others [15]. The

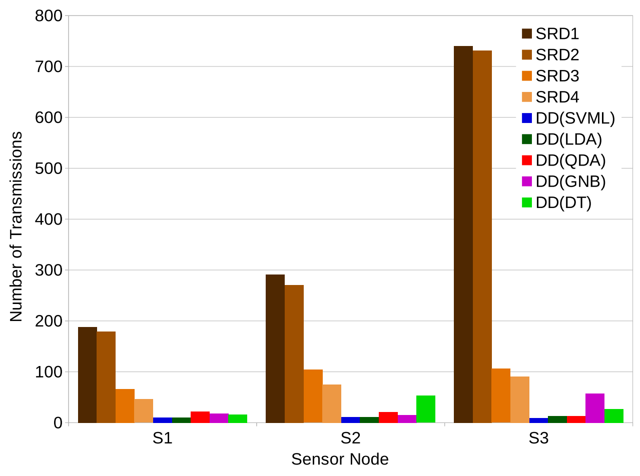

Figure 5: Number of transmissions encountered by S1, S2 and S3 while running different data reduction schemes.

penalty hyperparameter was fine-tuned in case of SVM (RBF) and Linear SVM for each of the experiments. For DT, the minimum samples split (*min_samples_split*) parameter was tuned to avoid overfitting on the training data. This parameter basically controls the amount of splitting in the DT. RF was used with 30 estimators and the tuned *min_samples_split* parameter. LDA, QDA and GNB do not require any hyperparameter tuning.

Table 2 presents the accuracy and the F1 score of each of the experiments described in the previous section. The best score of each experiment is shown in bold. For the CD and SRD cases where ML is applied at the edge, SVM with RBF kernel shines out more than the other algorithms suggesting a non-linear decision boundary to be a better fit for our data. Even though both LDA and Linear SVM are linear classifiers, LDA performs much better than the latter suggesting that the Gaussian assumption of the features holds well in this case. RF outperforms DT easily which is expected because of its ensemble nature. GNB performs the worst out of the seven algorithms which clearly implies that the feature independence assumption isn't valid for this particular application. If one were to observe the correlation between different features in Figure 4, this result seems plausible. In the DD case, the best result is given by Linear SVM and the numbers are comparable to the best of the other experiments. The hypothesis of the complete dataset performing better than the individual models as discussed in the previous section can be observed from the CD and DD rows of the table. Except for the GNB case which does not exploit correlations among features, ML algorithms on CD either equal or outperform the same on DD.

Figure 5 shows the number of transmissions experienced by each sensor node while running different data reduction schemes. Note that S4 has been omitted from the plot as there was no occupancy in zone 4 during our data collection phase which made training ML models for it irrelevant. Clearly, all the five ML algorithms of the DD case outperform the four SRD schemes for each sensor node by a huge margin. S3, being closer to the window, faces the most variation in the light data (hence the spike in SRD1 and SRD2 case for S3). SRD schemes are clearly affected by fast changing variations in data unlike the ML schemes. Linear SVM gives the least amount of transmissions (10, 11 and 9 out of 10,153 points for S1, S2 and S3). It may seem as if we can downsample the dataset more to make SRD match the Linear SVM in data reduction and while this can be achieved through trial and error, performance takes a big hit. For example, if we use $e_t = 2.0$, $e_h = 5.0$ and $e_l = 120$, we reduce the number of transmissions to almost what the Linear SVM gives but the F1 score drops down to 0.61 (SVM with RBF kernel). In general, for the occupancy estimation application, ML-based data reduction schemes show 18 to 82 times reduction in data transmissions as compared to Shewhart and 99.91% data transmission reduction overall in the best case while maintaining similar performance metrics.

5 CONCLUSIONS

In this paper, we described offloading strategies for five traditional ML algorithms to memory and computationally constrained embedded devices with experimental focus on the popular Atmel ATmega328P microcontroller. Based on this, we proposed a generic data reduction scheme for application-specific IoT networks wherein the sensor node(s) transmit only the inference and that too when it differs from the previously transmitted inference. In order to validate our proposed scheme, a sensor network testbed was deployed in a room for occupancy estimation and three different experiments were simulated on the collected dataset. It was experimentally shown that the ML-based data reduction schemes outperformed Shewhart-based schemes by reducing the number of transmissions by 18 to 82 times whilst imparting similar performance. Overall, 99.91% reduction in data transmission was achieved in the case of Linear SVM.

ACKNOWLEDGMENTS

The authors would like to thank Anish Shastri from Nokia Bell Labs, Ireland for useful and insightful discussions on various data reduction schemes.

REFERENCES

[1] Femi A. Aderohunmu, Giacomo Paci, Davide Brunelli, Jeremiah D. Deng, and Luca Benini. 2013. Prolonging the lifetime of wireless sensor networks using light-weight forecasting algorithms. In *2013 IEEE 8th International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. IEEE.
[2] Femi A. Aderohunmu, Giacomo Paci, Davide Brunelli, Jeremiah D. Deng, Luca Benini, and Martin Purvis. 2013. An Application-Specific Forecasting Algorithm for Extending WSN Lifetime. In *2013 IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE.
[3] Bakhtiar Qutub Ali, Niki Pissinou, and Kia Makki. 2008. Approximate replication of data using adaptive filters in Wireless Sensor Networks. In *2008 3rd International Symposium on Wireless Pervasive Computing*. IEEE, 365–369.
[4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory (COLT '92)*. ACM, 144–152. http://doi.acm.org/10.1145/130385.130401
[5] Gabriel Martins Dias, Boris Bellalta, and Simon Oechsner. 2016. A Survey About Prediction-Based Data Reduction in Wireless Sensor Networks. *ACM Comput. Surv.* 49, 3, Article 58 (Nov. 2016), 35 pages. http://doi.acm.org/10.1145/2996356
[6] Sridhar Gopinath, Nikhil Ghanathe, Vivek Seshadri, and Rahul Sharma. 2019. Compiling KB-sized Machine Learning Models to Tiny IoT Devices. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. ACM, 79–95. http://doi.acm.org/10.1145/3314221.3314597

[7] Yuichi Inagaki, Ryoichi Shinkuma, Takehiro Sato, and Eiji Oki. 2019. Prioritization of Mobile IoT Data Transmission Based on Data Importance Extracted From Machine Learning Model. *IEEE Access* 7 (2019), 93611–93620.

[8] Samantha Jacobs and Fernando Rios-Gutierrez. 2013. Novel Method for Using Q-learning in Small Microcontrollers. In *Proceedings of the 51st ACM Southeast Conference (ACMSE '13)*. ACM, Article 20, 4 pages. http://doi.acm.org/10.1145/2498328.2500065

[9] Ankur Jain, Edward Y. Chang, and Yuan-Fang Wang. 2004. Adaptive Stream Resource Management Using Kalman Filters. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*. ACM, 11–22. http://doi.acm.org/10.1145/1007568.1007573

[10] Petros Karvelis, Theofanis-Aristofanis Michail, Daniele Mazzei, Stefanos Petsios, Andrea Bau, Gabriele Montelisciani, and Chrysostomos Stylios. 2018. Adopting and Embedding Machine Learning Algorithms in Microcontroller for Weather Prediction. In *2018 International Conference on Intelligent Systems (IS)*. IEEE.

[11] Jarmo Lunden and Stefan Werner. 2015. Real-time smart metering with reduced communication and bounded error. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 326–331.

[12] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2003. The Design of an Acquisitional Query Processor for Sensor Networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*. ACM, 491–502. http://doi.acm.org/10.1145/872757.872817

[13] Mustafa K. Masood, Yeng Chai Soh, and Victor W.-C. Chang. 2015. Real-time occupancy estimation using environmental parameters. In *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[15] P. Perner. 2014. Machine Learning and Data Mining in Pattern Recognition. In *10th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*. Springer International Publishing.

[16] Anish Shastri, Vivek Jain, Sachin Chaudhari, Shailesh Singh Chouhan, and Stefan Werner. 2019. Improving Accuracy of the Shewhart-based Data-Reduction in IoT Nodes using Piggybacking. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 943–948.

[17] Tarek Sheltami, Muhammad Musaddiq, and Elhadi Shakshuki. 2016. Data compression techniques in Wireless Sensor Networks. *Future Generation Computer Systems* 64 (2016), 151 – 162. http://www.sciencedirect.com/science/article/pii/S0167739X16000285

[18] Adarsh Pal Singh, Vivek Jain, Sachin Chaudhari, Frank Alexander Kraemer, Stefan Werner, and Vishal Garg. 2018. Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes. In *2018 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 1–6.

[19] Liansheng Tan and Mou Wu. 2016. Data Reduction in Wireless Sensor Networks: A Hierarchical LMS Prediction Approach. *IEEE Sensors Journal* 16, 6 (2016), 1708–1715.