Distributed Machine Learning on Smart-Gateway Network Towards Real-Time Indoor Data Analytics

Huang, Hantao; Khalid, Rai Suleman; Yu, Hao

2017

Huang, H., Khalid, R. S., & Yu, H. (2017). Distributed Machine Learning on Smart-Gateway Network Towards Real-Time Indoor Data Analytics. In W. Pedrycz, & S.-M. Chen (Eds.), Data Science and Big Data: An Environment of Computational Intelligence (pp.231-263). Cham, Switzerland: Springer International Publishing.

<https://hdl.handle.net/10356/86077>

https://doi.org/10.1007/978-3-319-53474-9_11

© 2017 Springer International Publishing. This is the author created version of a work that has been peer reviewed and accepted for publication by Data Science and Big Data: An Environment of Computational Intelligence, Springer International Publishing. It incorporates referee's comments but changes resulting from the publishing process, such as copyediting, structural formatting, may not be reflected in this document. The published version is available at: [http://dx.doi.org/10.1007/978-3-319-53474-9_11].

Downloaded on 26 Apr 2023 13:06:28 SGT

Distributed Machine Learning on Smart-Gateway Network Towards Real-time Indoor Data Analytics

Hantao Huang, Rai Suleman Khalid and Hao Yu

Abstract Computational intelligence techniques are intelligent computational methodologies such as neural network to solve real-world complex problems. One example is to design a smart agent to make decisions within environment in response to the presence of human beings. Smart building/home is a typical computational intelligence based system enriched with sensors to gather information and processors to analyze it. Indoor computational intelligence based agents can perform behavior or feature extraction from environmental data such as power, temperature, and lighting data, and hence further help improve comfort level for human occupants in building. The current indoor system cannot address dynamic ambient change with a real-time response under emergency because processing backend in cloud takes latency. Therefore, in this chapter we have introduced distributed machine learning algorithms (SVM and neural network) mapped on smart-gateway networks. Scalability and robustness are considered to perform real-time data analytics. Furthermore, as the success of system depends on the trust of users, network intrusion detection for smart gateway has also been developed to provide system security. Experimental results have shown that with a distributed machine learning mapped on smart-gateway networks real-time data analytics can be performed to support sensitive, responsive and adaptive intelligent systems.

Hantao Huang

Nanyang Technological University, 50 Nanyang Avenue, Block S3.2, Level B2, Singapore 639798.
Tel.: (65) 6592 1844, Fax: (65) 6316 4416, E-mail: HHUANG013@e.ntu.edu.sg

Rai Suleman Khalid

Nanyang Technological University, 50 Nanyang Avenue, Block S3.2, Level B2, Singapore 639798.
Tel.: (65) 6592 1844, Fax: (65) 6316 4416, E-mail: RAIS0001@e.ntu.edu.sg

Hao Yu

Nanyang Technological University, 50 Nanyang Avenue, Block S3.2, Level B2, Singapore 639798.
Tel.: (65) 6592 1844, Fax: (65) 6316 4416, E-mail: haoyu@ntu.edu.sg

Keywords Computational Intelligence · Smart Home · Indoor Positioning · Distributed Machine Learning · Network Intrusion Detection · Support Vector Machine · Neural Network

1 Introduction

1.1 Computational Intelligence

Computational intelligence is the study of the theory, design and application of biologically and linguistically motivated computational paradigms [1–3]. Computational intelligence is widely applied to solve real-world problems which traditional methodologies can neither solve efficiently nor model feasibly. A typical computational intelligence based system can sense data from real-world, use this information to reason the environment and then performed desired actions. In a computational intelligent system such as smart building/home, collecting environmental data, reasoning the accumulated data and then selecting actions can further help to improve comfort level for human occupants. The intelligence of the systems comes from appropriate actions by reasoning the environmental data, which is mainly based on computational intelligence such as fuzzy logic and machine learning. To have a real-time response to the dynamic ambient change, a distributed system is preferred since a centralized system suffers long latency of processing in the back end [4]. Computational intelligence techniques (machine learning algorithms) have to be optimized to utilize the distributed yet computational resource limited devices.

1.2 Distributed Machine Learning

To tackle the challenge of high training complexity and long training time of machine learning algorithms, distributed machine learning is developed to utilize computing resources on sensors and gateway. Many recent distributed learning algorithms are developed for parallel computation across a cluster of computers by applying MapReduce software framework [5]. MapReduce shows a high capacity in handling intensive data and Hadoop is a popular implementation of MapReduce [6]. A prime attractive feature of MapReduce framework is its ability to take good care of data/code transport and nodes coordination. However, MapReduce services always have a high hardware requirement such as large processing memory in order to achieve good performance. However, IoT platforms, such as smart gateways, are with limited resources to support MapReduce operations.

Another kind of approaches is Message Passing Interface (MPI) based algorithms [7, 8]. MPI-based distributed machine learning has very low requirement for hardware and memory sources and it is very suitable for implementation and application in smart gateway environment. However, distribution schedulers for

traditional machine learning algorithms are naive and ineffective to utilize computational loads among nodes. Therefore, learning algorithms should be optimized to map on the distributed computational platform.

1.3 Indoor Positioning

GPS provides excellent outdoor services, but due to the lack of Line of Sight (LoS) transmissions between the satellites and the receivers, it is not capable of providing positioning services in indoor environment [9]. Developing a reliable and precise indoor positioning system (IPS) has been deeply researched as a compensation for GPS services in indoor environment. Wi-Fi based indoor positioning is becoming very popular these days due to its low cost, good noise immunity and low set-up complexity [10, 11]. Many WiFi-data based positioning systems have been developed recently for indoor positioning based on received signal strength indicator (RSSI) [12]. As the RSSI parameter can show large dynamic change under environmental change (such as obstacles) [13–15], the traditional machine-learning based WiFi data analytic algorithms can not adapt to the environment change because of the large latency. This is mainly due to the centralized computational system and the high training complexity [16], which will introduce large latency and also cannot be adopted on the sensor network directly. Therefore, in this chapter, we mainly focus on developing distributing indoor positioning algorithm targeting to computational resource limited devices.

1.4 Network Intrusion Detection

Any successful penetration is defined to be an intrusion which aims to compromise the security goals (i.e integrity, confidentiality or availability) of a computing and networking resource [17]. Intrusion detection systems (IDSs) are security systems used to monitor, recognize, and report malicious activities or policy violations in computer systems and networks. They work on the hypothesis that an intruder's behavior will be noticeably different from that of a legitimate user and that many unauthorized actions are detectable [18, 19]. Anderson et al. [17] defined the following terms to characterize a system prone to attacks:

- **Threat:** The potential possibility of a deliberate unauthorized attempt to access information, manipulate information or render a system unreliable or unusable.
- **Risk:** Accidental and unpredictable exposure of information, or violation of operations integrity due to malfunction of hardware or incomplete or incorrect software design.
- **Vulnerability:** A known or suspected flaw in the hardware or software design or operation of a system that exposes the system to penetration of its information to accidental disclosure.
- **Attack:** A specific formulation or execution of a plan to carry out a threat.

- **Penetration:** A successful attack in which the attacker has the ability to obtain unauthorized/undetected access to files and programs or the control state of a computer system.

The aim of cyber physical security techniques such as network intrusion detection system (NIDS) is to provide a reliable communication and operation of the whole system. This is especially necessary for network system such as Home Area Network (HAN), Neighborhood Area Network (NAN) and Wide Area Network (WAN) [20,21]. Network intrusion detection system can be placed at each network to detect network intrusion.

In this chapter, we will develop algorithms on distrusted gateway networks to detect network intrusions to provide system security.

1.5 Chapter Organizations

This chapter will be organized as follows. Firstly, we introduce a distributed computational platform on smart gateways for smart home management system in Section 2. Then, in Section 3, an indoor positioning system by support vector machine (SVM) and neural network is discussed and mapped on distributed gateway networks. In the following Section 4, a machine learning based network intrusion detection system (NIDS) is designed to provide system security. Finally, in Section 5, conclusion is drawn that distributed machine learning can utilize the limited computing resources and boost the performance of smart home.

2 Distributed Data Analytics Platform on Smart Gateways

2.1 Smart Home Management System

Smart Home Management System (SHMS) is an intelligent system built for residents to benefit from automation technology. By collecting environmental data including temperature, humidity and human activities, a system can react towards residents' best experience [22–25]. Fig. 1 depicts the basic components and working strategies in our SHMS test bed:

- **Smart gateways** to be the control center, harboring the ability in storage and computation. Our smart gateway will be BeagleBoard-xM.
- **Smart sensors** to collect environmental information on light intensity, temperature, humidity, and occupancy.
- **Smart sockets** to collect current information of home appliances.
- **Smart devices** with GUI to interact with users; residents have access to environmental information and can control home appliances through a smart phone or tablet.

To ensure high quality performance of SHMS, a robust indoor positioning system (IPS) is indispensable because knowledge about occupants of a building and their

Fig. 1: The overview of smart home management system

movements is essential [26]. Applications can include the scenarios when a resident comes back and enters his room, SHMS automatically powers on air conditioner, heater, humidifier and sets the indoor environment to suit the fitness condition; when nobody is in the house, the system turns off all appliances except for fridge and security system for energy saving issue.

2.2 Distributed Computation Platform

Fig. 2 shows our computation platform for real time data analytics. The major computation is performed on smart gateways in a distributed fashion. Data communication between gateways is performed through Wi-Fi using message passing interface (MPI). This distributed computation platform can perform real-time data analytics and store data locally for privacy purpose. Also, shared machine learning engine is developed in the smart gateway to perform real-time feature extraction and learning. Therefore, these learnt features can support indoor positioning services and provide network security protection.

An indoor positioning system (IPS) by WiFi-data consists of at least two hardware components: a transmitter unit and a measuring unit. Here we use smart gateways to collect WiFi signal emitted from other smart devices (phone, pad) of moving occupants inside the building. The IPS determines the positioning with WiFi-data analyzed from the smart gateway network [27]. The central unit in SHMS is BeagleBoard-xM as shown in Fig. 2b, which is also utilized in our positioning systems. In Fig. 2c, TL-WN722N wireless adapter is our Wi-Fi sensor for wireless signals capturing. BeagleBoard-xM runs Ubuntu 14.04 LTS with all the processing done on board, including data storage, Wi-Fi packet parsing, and positioning algorithm computation. TL-WN722N works in monitor mode, capturing packets ac-

Fig. 2: (a) Distributed computation platform (b) BeagleBoard xM (c) TL-WN722N (d) MAC frame format in Wi-Fi header field

cording to IEEE 802.11. They are connected with a USB 2.0 port on BeagleBoard-xM.

As depicted in Fig. 2d, Wi-Fi packet contains a header field (30 bytes in length), which contains information about Management and Control Address (MAC). This MAC address is unique to identify the device where the packet came from. Another useful header, which is added to the Wi-Fi packets when capturing frames, is the radio-tap header, which is added by the capturing device (TL-WN722N). This radio-tap header contains information about the RSSI, which reflects the information of distance [28].

Received Signal Strength Indicator (RSSI) is the input of indoor positioning system. It represents the signal power received at a destination node when signal was sent out from a source passing through certain space. RSSI has a relationship with distance, which can be given as:

$$RSSI = -K \log D + A; \quad (1)$$

where K is the slope of the standard plot, A is a fitting parameter and D is the distance [13]. So RSSI, as a basic measurement for distance, has been widely applied in Wi-Fi indoor positioning.

As described in Fig. 2, our SHMS is heavily dependent upon different entities communicating with each other over different network protocols (Wi-Fi, Ethernet, Zigbee). This in turn is prone to network intrusions. Some of the security violations that would create abnormal patterns of system usage include:

- Remote-to-Local Attacks (R2L): Unauthorized users trying to get into the system.

- User-to-Root Attacks (U2R): Legitimate users doing illegal activities and having unauthorized access to local superuser (root) privileges
- Probing Attacks: Unauthorized gathering of information about the system or network.
- Denial of Service (DOS) Attacks: Attempt to interrupt or degrade a service that a system provides to its intended users.

Therefore, the ability to detect a network intrusion is crucial for intelligent system to provide data and communication security.

3 Distributed Machine Learning based Indoor Positioning Data Analytics

3.1 Problem Formulation

The primary objective is to locate the target as accurate as possible considering the scalability and complexity.

Objective 1: Improve the accuracy of positioning subject to the defined area.

$$\begin{aligned} \min e &= \sqrt{(x_e - x_0)^2 + (y_e - y_0)^2} \\ s.t. \text{ label}(x_e, y_e) &\in \mathbf{T} \end{aligned} \quad (2)$$

where (x_e, y_e) is the system estimated position belongs to the positioning set $\mathbf{T}$ and (x_0, y_0) is the real location coordinates. Therefore, a symbolic model based positioning problem can be solved using training set Ω to develop neural-network.

$$\Omega = \{(s_i, t_i), i = 1, \dots, N, s_i \in R^n, t_i \in \mathbf{T}\} \quad (3)$$

where N represents the number of datasets and n is the number of smart gateways, which can be viewed as the dimension of the signal strength space. s_i is the vector containing RSSI values collected in i_{th} dataset, $t_i \in \{-1, 1\}$ is a discrete value and denoted as a label to represent the indoor positioning coordinates. Note that $\mathbf{T} \in \{-1, 1\}$ labels the physical position and by changing the sequence of -1 and 1 , different labels can be represented. The more labels are used, the more accurate the positioning service is.

Objective 2: Reduce the training time of machine learning on Hardware BeagleBoard-xM. To distribute training task on gateways with n number, the average training time should be minimized to reflect the reduced complexity on such gateway system.

$$\begin{aligned} \min \frac{1}{n} \sum_{i=1}^n t_{train,i} \\ s.t. \text{ } e < \epsilon \end{aligned} \quad (4)$$

where $t_{train,i}$ is the training time on i_{th} smart gateway, e is the training error and ϵ is the tolerable maximum error.

3.2 Indoor Positioning By Distributed SVM

Support vector machine (SVM) is one robust machine learning algorithm and can be viewed as a special form of neural network [29]. Due to its reliability and high accuracy in classification, it is one of the most popular algorithms in positioning. The basic idea behind SVM in indoor positioning is to build a classification model based on a training set [30]. To obtain the decision function in the classification problem, LIBSVM [31], one of the most popular SVM solver, is chosen for our multi-category positioning problem.

For indoor positioning application, different positioning zones mean different classes, so multi-category classification is required here. Multi-category classification in LIBSVM uses one-versus-one (1-v-1) SVM, i.e., if there are k classes, the solution follows binary problems solved by 1 vs. 2, 1 vs. 3, ..., 1 vs. k , 2 vs. 3, ..., 2 vs. k , ..., $k-1$ vs. k , with a total number of $k(k-1)/2$. This will generate $k(k-1)/2$ decision functions of where $F: R^n \rightarrow \{C_i, C_j\}, i, j = 1, 2, \dots, k, i \neq j$ where C_i means i_{th} class.

3.2.1 Workload Scheduling For DSVM

Experimental results show that iteration time, the time spent on generating binary decision functions, is the major part of time consumed in training (more than 95%). The balanced training time can be expressed as (5):

$$\sum_{i=1}^n t_{train,i} = \sum_{p=1}^{K(K-1)/2} t_p \quad (5)$$

where t_p means the time consumed for subtask p and $t_{train,i}$ is the training time on i_{th} smart gateway,

Thus, we can ease the load on a single gateway by simply distributing iteration tasks. This section will focus on evenly distributing all the sub tasks onto multiple gateways for processing in order to improve performance. In order to elaborate our task distribution strategies with more details, we first introduce two necessary definitions:

Definition 1: The task workload is the factor to measure each task time in computation. The higher workload a task has, the more computational time it needs for this sub-task. The workloads for all sub-tasks are stored in a matrix $\mathbf{L}$, where a L_p stores the workload of sub-task p . As the computation time in SVM training is quadratic in terms of the number of training instances, task workloads are represented by square of dataset size of sub-training tasks.

Definition 2: The decision matrix $\mathbf{B}$ is an $M \times P$ matrix to define whether sub-task p is allocated to gateway m , where M is the number of gateways and P is the number of sub-tasks.

$$\mathbf{B}_{m,p} = \begin{cases} 1, & \text{sub task } p \text{ is with gate } m \\ 0, & \text{sub task } p \text{ not with gate } m \end{cases} \quad (6)$$

Based on the task workload distribution and decision matrix, we can have the final workload allocation on each gate:

$$\begin{aligned}\mathbf{G}_m &= \mathbf{B}_m \mathbf{L}, \quad m = 1, 2, \dots, M \\ &= [\mathbf{B}_{m,1}, \mathbf{B}_{m,2}, \dots, \mathbf{B}_{m,p}] [L_1, L_2, \dots, L_p]^T\end{aligned}\quad (7)$$

where $\mathbf{L}$ represents all sub-tasks stored, where L_p stores the workload of sub-task p . $\mathbf{B}_m$ represents the decision matrix for gateway m and $\mathbf{G}_m$ is the total workload allocated to m_{th} gateway. With these definitions, an optimization WL-DSVM model can be described as (8)

$$\min[\max(\mathbf{G}_m) + \sqrt{\sum_{m=1}^M (\mathbf{G}_m - \overline{\mathbf{G}_m})^2 / M}] \quad (8)$$

$$s.t. \begin{cases} \mathbf{G}_m = \mathbf{B}_m \mathbf{L}, & m = 1, 2, \dots, M \\ \mathbf{B}_{m,p} = 0 \text{ or } 1, & m = 1, 2, \dots, M; p = 1, 2, \dots, P \end{cases} \quad (9)$$

The conventional allocation scheme in [30] is following a computational sequence, and ignoring the difference in computation complexity between sub-problems. However, we take workload of each sub-task into consideration and reorder task allocation. The distribution schemes comparison is described in Fig. 3. If we have a 5-class problem, which will generate 10 sub-tasks (or binary-class classifications), sequential DSVM (SQ-DSVM) will allocate the sub-tasks between 2 gateways by following the computational sequence as shown in Fig. 3: tasks 1-4 allocated to gateway 1; tasks 5-7 allocated to gateway 2; tasks 8-9 allocated to gateway 1; tasks 10 allocated to gateway 2. To achieve an even distribution, we need to rearrange the sub-tasks according to workload distribution and allocate them in an even manner to gateways. In order to realize it, we propose an allocation scheme based on a greedy rule that always allocates the heaviest workload to the gateway with the lightest workload.

As a summary, the pseudo-code and initial settings in the proposed workload-based DSVM are shown in Algorithm 1. Initial values are presented as input. In line 1 we sort the workloads of gates from large ones to small ones. The updated sub-task index in l_s is with line 2. In line 3 we randomize gate index in case that some gates are always allocated more loads than others. Loop line 4-9 is the main part of Workload-based WL-DSVM computation load allocation. Finally in line 8, decision matrix will be updated and the iteration moves to the next loop.

3.2.2 Working Flow for DSVM-Based IPS

With a distributed fashion applied in support vector machine (SVM), an ideal result is that working load can be distributed and training time can be reduced to:

$$T_{train} = t_{run} + \sum_{p=1}^P t_p / P \quad (10)$$

Fig. 3: Working scheduling schemes of sequential based distributed SVM (SQ-DSVM) and workload based distributed SVM (WL-DSVM)

For predicting phase, as only a very short time is needed to predict a user's position, it's not necessary to distribute one single prediction task among different gateways. However, for plenty of incoming RSSI arrays, to predict position of all data on one gateway will have the working load unevenly allocated and there will be one node put under too much load. The solution is to introduce a distributed data storage system. RSSI series stored in m_{th} gateway can be determined using the following formula:

$$S_i = \{s | RSSI_m = \max(s), RSSI_{m'} \neq \max(s), m' < m\} \quad (11)$$

which means the maximum RSSI value of the data stored in m_{th} gateway is exactly $RSSI_m$; s represents a RSSI array. But if there existing case of $RSSI_1 = RSSI_2$, assuming that we have stored the dataset in gateway 1, equation (11) will avoid storing the same dataset in gateway 2 again ($m = 2, m' = 1$ as described in (11)). As such, working flow of proposed DSVM indoor positioning can be described as Fig. 4.

3.2.3 Experiment Result

Algorithm 1: WL-DSVM workload allocation

Input : Sub-task number n
 Computational load vector for each sub-task L
 Index vector for each sub-task $l_s = [1, 2, \dots, P]$
 Working load vector for each gate $G = 0$
 Index vector for each gateway $l_g = [1, 2, \dots, M]$
 Decision Matrix $B = 0$

Output: Working load allocation for each sub-task B

```

1 SortFromLargeToSmall(a)
2 Update order in  $l_s$  based on  $L$ 
3 RandGateIndex ( $I_g$ )
4 for  $p \leftarrow 1$  to  $P$  //load allocation for each gate do
5   SortFromSmallToLarge( $G$ )
6   Update order in  $I_g$  based on  $G$ 
7   Update gateway load  $G_1 \leftarrow G_1 + L_p$ 
8   Update decision matrix  $B(l_{g,1,p}) \leftarrow 1$ 
9 end

```

Fig. 4: Working scheduling schemes of SQ-DSVM and WL-DSVM

System and Simulation Setup Before verifying the effectiveness of distributed support vector machine, we test the computational capacity of PC and gateway with $4800 \times \text{Dup}$ (Dup=1 to 10 means size of datasets are $4800 \times \text{Dup}$, separately, the duplicate count in Fig. 5) datasets for 8 classes, 70% for training, 30% for testing; each dataset has 5 features, i.e. RSSI1, RSSI2, ..., RSSI5. Fig. 5 shows the time for training/testing on BeagleBoard-xM (BB)/PC. As is shown in Fig. 5, training on smart gateway (AM37x 1GHz ARM processor) takes a rather long time compared with PC (Intel[®] CoreTM i5 CPU 650 3.20 GHz), and the divergence is about 10 times. Conversely, the predicting time is low enough that it can be quickly handled on the board.

To verify the effectiveness of distributed support vector machine (DSVM) in indoor positioning system (IPS), we performed tests with our indoor environment.

Fig. 5: Runtime comparison for PC and BB based SVMs

For comparison, we implemented conventional SVM algorithm on PC (Baseline 1) and BeagleBoard-xM (Baseline 2) separately, sequential based DSVM (SQ-DSVM) on board (Baseline 3), and finally workload-based DSVM (WL-DSVM) on board (Baseline 4).

Baseline 1: Centralized SVM algorithm. In this method, data (RSSI arrays and position) are collected by different gateways but data processing is performed on a single PC, denoted as SVM-PC.

Baseline 2: Centralized SVM algorithm. In this method, data are collected by different gateways and data processing is performed on Gateway 1, denoted as SVM-BB.

Baseline 3: SQ-DSVM algorithm. In this method, data collection and algorithm computation are both paralleled among 5 gateways. In this method, training phase is performed with sequential DSVM, it is denoted as SQ-DSVM-BB.

Baseline 4: WL-DSVM algorithm. In this method, data collection and algorithm computation are both paralleled among 5 gateways. In this method, training phase is performed with workload-based DSVM, denoted as WL-DSVM-BB.

Indoor test-bed environment for positioning is presented in Fig. 6, with total area being about $80 m^2$ (8 m at width and 10 m at length) separated into 48 regular blocks, each block represents a research cubicle in the lab, and the cubicles are the positioning areas in this chapter. 5 gateways, with 4 locations at 4 corners of the map, 1 in the center of the map, are set up for experiment.

To quantify our environment setting, here the positioning accuracy is defined as r , representing radius of target area. It is generated from $S = \pi r^2$, where S is the square of the whole possible positioning area.

Besides, positioning precision is defined as the probability that the targets are correctly positioned within certain accuracy. The definition is as follow:

Fig. 6: An example of building floor with position tracking

$$Precision = \frac{N_{pc}}{N_p} \quad (12)$$

where N_{pc} is the number of correct predictions and N_p is the number of total predictions.

Performance Comparison To verify the effectiveness of distributed support vector machine (DSVM) in indoor positioning system (IPS), in this part, performance comparison between WL-DSVM and other SVMs will be performed within our indoor environment. For comparison, we implemented centralized SVM algorithm on PC (SVM-PC) and BeagleBoard-xM (SVM-BB) separately, and distributed SVMs are tested with workload-based DSVM on board (WL-DSVM-BB) and sequential DSVM on board (SQ-DSVM-BB).

Table 1 (Dup =1,2,...,10 means size of datasets are $4800 \times \text{Dup}$, separately) mainly elaborates the advantages of distributed machine learning over centralized data analytics. (1) Without a distributed fashion, centralized SVM on board only shows about 1/8 in computational ability of PC. (2) With DSVM on 5 gateways, improvement of computational efficiency in runtime can be 2.5x-3.5x of SVM-BB. (3) WL-DSVM-BB shows a higher efficiency in runtime than SQ-DSVM-BB; (4) Increased size of dataset for training doesn't show an increase in relative time with distributed machine learning; it means that computational load is much more time consuming than data communication and needs more attention.

Fig. 7 shows the running time comparison of each BeagleBoard with WL-DSVM-BB and SQ-DSVM-BB methods. The total time of each BeagleBoard includes the training time, communication time and waiting time for others. It clearly shows that WL-DSVM-BB has relative fair load distribution. Therefore, WL-DSVM-BB performs a higher training efficiency with total time 679s ($\max(G_m)$ in (8)), comparing with total time 860s on SQ-DSVM-BB, achieving an improvement of 27% for the case of Dup =10.

Table 1: Training Time Comparison Of Among Different SVMs

Dup	Time consumed (seconds)							
	SVM-PC		SVM-BB		SQ-DSVM-BB		WL-DSVM-BB	
1	3.30	1×	25.78	7.8×	11.58	3.5×	9.41	2.9×
2	13.23	1×	99.09	7.5×	44.26	3.3×	36.78	2.8×
3	29.75	1×	221.84	7.5×	95.38	3.2×	79.85	2.7×
...	...	...	...	...	...	...	...	...
7	137.44	1×	1101.98	8.0×	451.18	3.3×	362.84	2.6×
8	179.33	1×	1452.87	8.1×	611.85	3.4×	509.18	2.8×
9	228.08	1×	1842.62	8.1×	770.86	3.4×	614.57	2.7×
10	266.68	1×	2195.79	8.2×	859.90	3.2×	679.27	2.5×

Fig. 7: Runtime comparison on each gateway for WL-DSVM and SQ-DSVM. Total time includes the training time, communication time and waiting time for other beagleboards

WL-DSVM-BB only needs $2.5\times$ training time of PC, and results in $3.2\times$ improvement in training time when compared with centralized SVM on board (SVM-BB), which means WL-DSVM has made sense in reducing the working load of a single node, which is useful when gateway nodes are in a large amount, such as hundreds of gateways, which will be very promising in real-time positioning for a changing environment. Conventional way in training phase is off-line training on PC and then sending the training predictor to sensor nodes. But with an ability to efficiently compute training phase on board, real-time data analysis can be performed on board so that we can get rid of server that requires extra cost and implementation.

In order to test the positioning precision improvement with WL-DSVM, we simulate an indoor environment where RSSI values vary according to Gaussian distribution every half an hour. WL-DSVM can update its predictor automatically while SVM-BB applies the initially generated predictor. This is a reasonable setting since WL-DSVM is much faster to perform training than SVM-BB. Due to the long training time SVM-BB is not likely to perform online updates while WL-DSVM

Fig. 8: Indoor positioning accuracy & precision by WL-DSVM

Fig. 9: Positioning precision comparison of changing environment for WL-DSVM and SVM-BB

is favorable to perform online updates. Here, we take accuracy of 2.91m for an example. Results in Fig. 9 show that WL-DSVM can maintain the precision of prediction, while SVM shows a decreasing and unstable performance of precision. It is because WL-DSVM will update the predictor whenever environment changes. But traditionally, training phase is only done once so that the precision will decrease with a changing environment. Indoor positioning accuracy and its precision by WL-DSVM are shown in Fig. 8.

In conclusion, instead of dealing with data storage and analytics on one central server, the proposed distributed real-time data analytics is developed on networked gateways with limited computational resources. By utilizing the distributed support vector machine (DSVM) algorithm, the data analytics of real-time RSSI values of

Fig. 10: Soft-voting based distributed-neural-network with 2 sub-systems

Wi-Fi data can be mapped on each individual gateway. The experimental results have shown that the proposed WL-DSVM can achieve a 3.2x improvement in runtime in comparison to a single centralized node and can achieve a performance improvement of 27% in runtime in comparison to conventional DSVM.

3.3 Indoor Positioning By Distributed-neural-network

In this section, we introduce a low computational complexity machine-learning algorithm that can perform WiFi- data analytics for positioning on smart gateway network. A distributed-neural-network (DNN) machine learning algorithm is introduced with the maximum posteriori probability based soft-voting. Experiment results have shown significant training and testing speed-up comparing to SVM.

3.3.1 Machine Learning Algorithm On Gateway

Single-hidden-layer Neural-network Our neural-network with two sub-systems is shown as Fig. 10, which is inspired by extreme learning machine and compressed sensing [32], [33]. Unlike previous work [14], the input weight is only connecting nearby hidden nodes. The input weight in our proposed neural-network is connected to every hidden node and is randomly generated independent of training data. [34] Therefore, only the output weight is calculated from the training process. Assume there are N arbitrary distinct training samples $\mathbf{X} \in \mathbf{R}^{N \times n}$ and $\mathbf{T} \in \mathbf{R}^{N \times m}$, where $\mathbf{X}$ is training data representing scaled RSSI values from each gateway and $\mathbf{T}$ is the training label indicating its position respectively. In our indoor positioning cases, the relation between the hidden neural-node and input training data is additive as

$$\text{preH} = \mathbf{X}\mathbf{A} + \mathbf{B}, \quad \mathbf{H} = \frac{1}{1 + e^{-\text{preH}}} \quad (13)$$

where $\mathbf{A} \in \mathbf{R}^{n \times L}$ and $\mathbf{B} \in \mathbf{R}^{N \times L}$. $\mathbf{A}$ and $\mathbf{B}$ are randomly generated input weight and bias formed by a_{ij} and b_{ij} between $[-1, 1]$. $\mathbf{H} \in \mathbf{R}^{N \times L}$ is the result from

Algorithm 2: Learning Algorithm for Single Layer Network

Input : Training Set $(x_i, t_i), x_i \in \mathbf{R}^n, t_i \in \mathbf{R}^m, i = 1, \dots, N$, activation function $H(a_{ij}, b_{ij}, x_i)$, maximum number of hidden neural node L_{max} and accepted training error ϵ .

Output: Neural-network output weight β

- 1 Randomly assign hidden-node parameters
- 2 $(a_{ij}, b_{ij}), a_{ij} \in \mathbf{A}, b_{ij} \in \mathbf{B}$
- 3 Calculate the hidden-layer pre-output matrix $\mathbf{H}$
- 4 $\mathbf{preH} = \mathbf{XA} + \mathbf{B}, \mathbf{H} = 1/(1 + e^{-\mathbf{preH}})$
- 5 Calculate the output weight
- 6 $\beta = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T}$
- 7 Calculate the training error *error*
- 8 $error = \|\mathbf{T} - \mathbf{H}\beta\|$
- 9 **if** $(L \leq L_{max} \text{ and } e > \epsilon)$ **then**
- 10 Increase number of hidden node
- 11 $L = L + 1$, repeat from Step 1
- 12 **end**

sigmoid function for activation. In general cases, the number of training data is much larger than the number of hidden neural nodes (i.e. $N > L$), to find the output weight β is an overdetermined system. Therefore, estimating the output weight is equivalent to minimize $\|\mathbf{T} - \mathbf{H}\beta\|$, the general solution can be found as

$$\beta = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T}, \mathbf{H} \in \mathbf{R}^{N \times L} \quad (14)$$

where $\beta \in \mathbf{R}^{L \times m}$ and m is the number of symbolic classes. $(\mathbf{H}^T \times \mathbf{H})^{-1}$ exists for full column rank of $\mathbf{H}$ [32]. However, such method is computationally intensive. Moreover, as the number of hidden neural nodes can not be explicit from the training data to have small training error, [35] suggests to increase the number of hidden neural node L during the training stage, which will reduce the training error but at the cost of increasing computational cost and required memory for neural-network. Therefore, an incremental solution for (14) is needed to adjust the number of hidden node L with low complexity. The algorithm of single hidden layer neural-network is summarized in Algorithm 2.

Incremental Least-square Solver The key difficulty for solving training problem is the least square problem of minimizing $\|\mathbf{T} - \mathbf{H}\beta\|$. This could be solved by using SVD, QR and Cholesky decomposition. The computational cost of SVD, QR and Cholesky decomposition is $O(4NL^2 - \frac{4}{3}L^3)$, $O(2NL^2 - \frac{2}{3}L^3)$ and $O(\frac{1}{3}L^3)$ respectively [36]. Therefore, we use Cholesky decomposition to solve the least square problem. Moreover, its incremental and symmetric property reduces the computational cost and saves half memory required [36]. Here, we use $\mathbf{H}_L$ to represent the matrix with L number of hidden neural nodes ($L < N$), which decomposes the symmetric positive definite matrix $\mathbf{H}^T \mathbf{H}$ into

$$\mathbf{H}_L^T \mathbf{H}_L = \mathbf{Q}_L \mathbf{Q}_L^T \quad (15)$$

Algorithm 3: Incremental L_2 norm solution

Input : Activation matrix $\mathbf{H}_L$, target matrix $\mathbf{T}$ and number of hidden nodes L
Output: Neural-network output weight β

```

1 for  $l \leftarrow 2$  to  $L$  do
2   Calculate new added column
3    $\mathbf{v}_l \leftarrow \mathbf{H}_{l-1}^T h_l$ 
4    $g \leftarrow h_l^T * h_l$ 
5   Calculate updated Cholesky matrix
6    $\mathbf{z}_L \leftarrow \mathbf{Q}_{L-1}^{-1} \mathbf{v}_L$ ,  $p \leftarrow \sqrt{g - \mathbf{z}_L^T \mathbf{z}_L}$ 
7   Form new Cholesky Matrix  $\mathbf{Q}_L \leftarrow \begin{pmatrix} \mathbf{Q}_{L-1} & 0 \\ \mathbf{z}_L^T & p \end{pmatrix}$ 
8   Calculate output weight using forward and backward substitution
9    $\mathbf{Q}_L \mathbf{Q}_L^T \beta \leftarrow \mathbf{H}_L^T \mathbf{T}$ 
10 end

```

where $\mathbf{Q}_L$ is a low triangular matrix and T represents transpose operation of the matrix.

$$\begin{aligned} \mathbf{H}_L^T \mathbf{H}_L &= [\mathbf{H}_{L-1} \ h_L]^T [\mathbf{H}_{L-1} \ h_L] \\ &= \begin{pmatrix} \mathbf{H}_{L-1}^T \mathbf{H}_{L-1} & \mathbf{v}_L^T \\ \mathbf{v}_L^T & g \end{pmatrix} \end{aligned} \quad (16)$$

where h_L is the new added column by increasing the size of L , which can be calculated from (13). The Cholesky matrix can be expressed as

$$\begin{aligned} \mathbf{Q}_L \mathbf{Q}_L^T &= \begin{pmatrix} \mathbf{Q}_{L-1} & 0 \\ \mathbf{z}_L^T & p \end{pmatrix} \begin{pmatrix} \mathbf{Q}_{L-1}^T & \mathbf{z}_L \\ 0 & p \end{pmatrix} \end{aligned} \quad (17)$$

As a result, we can easily calculate the $\mathbf{z}_L$ and scalar p for Cholesky factorization as

$$\mathbf{Q}_{L-1} \mathbf{z}_L = \mathbf{v}_L, \quad p = \sqrt{g - \mathbf{z}_L^T \mathbf{z}_L} \quad (18)$$

where $\mathbf{Q}_{L-1}$ is the previous Cholesky decomposition result and $\mathbf{v}_L$ is known from (16), which means we can continue to use previous factorization result and update only according part. Algorithm 3 gives details on each step since $l \geq 2$. Please note when $l = 1$, $\mathbf{Q}_1$ is a scalar and equals to $\sqrt{H_1^T H_1}$. Such method will greatly reduce computational cost and allow the online training on smart gateway for positioning.

3.3.2 Distributed-neural-network with Soft-voting

Distributed-neural-network for Indoor Positioning The working flow of the distributed-neural-network (DNN) is described as Fig. 11. Environmental Wi-Fi signal is received from the Wi-Fi Adapter and through Wi-Fi parsing the data with MAC address and RSSI of each Wi-Fi adapter is stored. Such data is sent to gateway for training with label first. Please note that the training process is on the gateway. As

Fig. 11: Working flow of distributed-neural-network indoor positioning system

we mentioned in Section 3.3.1, a single layer forward network (SLFN) is trained. A small data storage is required to store trained weight for the network. In the real time application, the same format data will be collected and sent into the well trained network to locate its position. In Fig. 11, the block for soft-voting is through message passing interface (MPI) protocols to collect all the testing result from each SLFN and soft-voting is processed in the central gateway. Note that n gateways together can form one or several SLFNs based on the accuracy requirement.

Soft-voting As we have discussed in section 3.3.1, the input weight and bias $\mathbf{A}, \mathbf{B}$ are randomly generated, which strongly supports that each SLFN is an independent expert for indoor positioning. Each gateway will generate posteriori class probabilities $P_j(c_i|\mathbf{x}), i = 1, 2, \dots, m, j = 1, 2, \dots, N_{slfn}$, where $\mathbf{x}$ is the received data, m is the number of classes and N_{slfn} is the number of sub-systems for single layer network deployed on smart gateway. During the testing process, the output of single layer forward network (SLFN) will be a set of values $y_i, i = 1, 2, \dots, m$. Usually, the maximum y_i is selected to represent its class i . However, in our case, we scale the training and testing input between $[-1, 1]$ and target labels are also formed using a set of $[-1, -1, \dots, 1, \dots, -1]$, where the only 1 represents its class and the target label has length m . The posteriori probability is estimated as

$$P_j(c_i|\mathbf{x}) = (y_i + 1)/2, j = 1, 2, \dots, N_{slfn} \quad (19)$$

A loosely stated objective is to combine the posteriori of all sub-systems to make more accurate decisions for the incoming data $\mathbf{x}$. Under such case, information theory suggests to use a cross entropy (Kullback-Leibler distance) criterion [37], where we may have two possible ways to combine the decisions (Geometric average rule and Arithmetic average rule). The geometric average estimates can be

calculated as

$$P(c_i) = \prod_{j=1}^{N_{slfn}} P_j(c_i|\mathbf{x}), \quad i = 1, 2, \dots, m \quad (20)$$

and the arithmetic average estimate is shown as

$$P(c_i) = \frac{1}{N_{slfn}} \sum_{j=1}^{N_{slfn}} P_j(c_i|\mathbf{x}), \quad i = 1, 2, \dots, m \quad (21)$$

where $P(c_i)$ is the posteriori probability to choose class c_i and will select the maximum posteriori $P(c_i)$ for both cases. In this chapter, we use arithmetic average as soft-voting of each gateway since [37] indicates that geometric average rule works poorly when the posteriori probability is very low. This may happen when the object to locate is far away from one gateway and its RSSI is small with low accuracy of positioning. The final decision is processed at the central gateway to collect the voting value from each sub-systems on other gateways. Such soft-voting will utilize the confidence of each sub-system and avoid the prerequisite that each sub-system maintains accuracy of more than 50 % for hard-voting.

Table 2: Experimental set-up parameters

Parameter	Value
Training Data Size	18056
Testing Data Size	2000
Data Dimension	5
Number of labels	48
No. of Gateway	5
Testing area	80m ²

3.3.3 Experimental Results

Experiment Setup Indoor test-bed environment for positioning is presented in Fig. 6, which is the same as Section 3.2.3. The summary for the experiment set-up is shown in Table 2. To avoid confusion, we use DNN to represent distributed neural network and SV-DNN represents soft-voting based DNN.

Real-time Indoor Positioning Results The result of the trained neural forward network is shown as Fig. 12 and Fig. 13. The training time can be greatly reduced by using incremental Cholesky decomposition. This is due to the reduction of least square complexity, which is the limitation for the training process. As shown in Fig. 12, training time maintains almost constant with increasing number of neural nodes when the previous training results are available. Fig. 13 also shows the increasing accuracy under different positioning scales from 0.73m to 4.57m. It also shows that increasing the number of neural nodes will increase the performance to certain accuracy and maintains almost flat at larger number of neural nodes.

Table 3: Comparison table with previous works

System/Solution	Precision
Proposed DNN	58% within 1.5m, 74% within 2.2m and 87% within 3.64m
Proposed SV-DNN	62.5% within 1.5m, 79% within 2.2m and 91.2% within 3.64m
Microsoft RADAR [38]	50% within 2.5m and 90% within 5.9m
DIT [39]	90% within 5.12m for SVM; 90% within 5.40m for MLP
Ekahau [40]	5 to 50m accuracy (indoors)
SVM	63% within 1.5m, 80% within 2.2m and 92.6% within 3.64m

Table 4: Performance precision with variations on proposed DNN with soft-voting

	Testing time (s)	Training time (s)	0.73m	1.46m	2.19m	2.91m	3.64m	4.37m	5.1m	No. of Nodes
SVM	9.7580	128.61	31.89%	63.26%	80.25%	88.54%	92.58%	94.15%	94.71%	N.A.
Acc. & Var.			0.530	0.324	0.394	0.598	0.536	0.264	0.0975	
DNN	0.1805	1.065	23.94%	57.78%	74.14%	82.61%	87.22%	90.14%	91.38%	100
Acc. & Var.			1.2153	0.0321	0.1357	0.2849	0.0393	0.0797	0.0530	
SV-DNN (2)	0.1874	2.171	29.36%	61.23%	77.20%	86.24%	90.25%	92.19%	93.14%	2 Sub-systems Each 100
Acc. & Var.			0.358	0.937	0.526	0.517	0.173	0.173	0.124	
SV-DNN (3)	0.1962	3.347	30.52%	62.50%	79.15%	87.88%	91.20%	92.92%	94.08%	3 Sub-systems Each 100
Acc. & Var.			0.325	1.952	0.884	1.245	0.730	0.409	0.293	

Performance Comparison In Table 3, we can see that although single layer network cannot perform better than SVM but it outperforms other positioning algorithms proposed in [38–40]. Moreover, by using maximum posteriori probability based soft-voting, SV-DNN can be very close to the accuracy of SVM. Table 4 shows the detailed comparisons between proposed DNN positioning algorithm with SVM. Please note that the time reported is the total time for training data size 18056 and testing data size 2000. It shows more than 120x training time improvement and more than 54x testing time saving for proposed SLFN with 1 sub-network comparing to SVM. Even adding soft-voting with 3 sub-networks, 50x and 38x improvement in testing and training time respectively can be achieved. Please note that for fair training and testing time comparison, all the time is recorded using Ubuntu 14.04 LTS system with core 3.2GHz and 8GB RAM. Variances of the accuracy is also achieved by 5 repetitions of experiments and the reported results are the average values. We find that the stability of proposed DNN is comparable to SVM.

Fig. 12: Training time for SLFN by Incremental Cholsky decomposition

Fig. 13: Testing Accuracy under different positioning scale

Fig. 14: Error Zone and accuracy for indoor positioning by distributed neural network (DNN)

Moreover, the testing and training time do not increase significantly with new added subnetworks. Please note that SVM is mainly limited by its training complexity and binary nature where one-against-one strategy is used to ensure accuracy with a cost of building $m(m-1)/2$ classifier and m is the number of classes. Fig. 14 shows the error zone of proposed SV-DNN.

In conclusion, this section proposes a computationally efficient data analytics by distributed-neural-network (DNN) based machine learning with application for indoor positioning. It is based on one incremental L_2 -norm based solver for learning collected WiFi-data at each gateway and is further fused for all gateways in the network to determine the location. Experimental results show that with 5 distributed gateways running in parallel for a $80m^2$ space, the proposed algorithm can achieve 50x and 38x improvement on testing and training time respectively when compared to support vector machine based data analytics with comparable positioning precision.

4 Distributed Machine Learning based Network Intrusion Detection System

In this section we propose to use the distributed-neural-network (DNN) method as described in section 3.3.1 for the network intrusion detection system (NIDS). We also use the same soft voting technique as described in section 3.3.2 to achieve an improved accuracy.

4.1 Problem Formulation and Analysis

In machine learning approach for NIDS, the detection for intrusion can be considered as a binary classification problem, distinguishing between normal and at-

Fig. 15: NIDS Flow Based On Distributed Machine Learning On Smart-gateways

tack instances. In the similar way, intrusion can also be considered as a multi-class classification problem to detect different attacks. We can use supervised, semi-supervised or unsupervised machine learning approach to achieve the objective. In this section we use a supervised machine learning approach based on single hidden layer neural-network [32] for intrusion detection. Fig. 15 shows an overview of steps involved in binary NIDS. As such our main objectives are :

Objective 1: Achieve an overall high accuracy, high detection rate, a very low false alarm rate. We define the following terms to mathematically formulate our objective.

1. *False Positives (FP)*: Number of normal instances which are detected as intrusions.
2. *False Negatives (FN)*: Number of intrusion instances which are detected as normal.
3. *True Positives (TP)*: Number of correctly detected intrusion instances.
4. *True Negatives (TN)*: Number of correctly detected normal instances.

Several statistical measures are used to measure the performance of machine learning algorithms for Binary NIDS. Specifically following measures are used to characterize the performance.

1. *Recall*: is a measure of detection rate of the system to detect attacks and is defined as :

$$Recall = \frac{TP}{TP + FN} \times 100 \quad (22)$$

2. *False Positive Rate (FP)*: gives a measure of false positive rate i.e., normal instances being classified as intrusions

$$FP = \frac{FP}{FP + TN} \times 100 \quad (23)$$

3. *Precision* : Precision is a measure of predicted positives which are actual positives.

$$Precision = \frac{TP}{TP + FP} \times 100 \quad (24)$$

4. *F-Measure* : F-Measure is a metric that gives a better measure of accuracy of an IDS. It is a harmonic mean of precision and recall.

$$F - Measure = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \times 100 \quad (25)$$

5. *Matthews Correlation Coefficient (MCC)* : MCC measures the quality of binary classification. It represents values in the range -1 to +1. A value of +1 represents 100 % prediction, -1 represents 0 % prediction. A value of 0 represents no better prediction than random prediction.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (26)$$

6. *Overall Accuracy*: Overall accuracy is defined as a ratio of TP and TN to the total number of instances

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN} \times 100 \quad (27)$$

Objective 2: Reduce the training complexity for intrusion detection modules at various network layers in the smart grid so that the model can be quickly updated. To distribute training task on gateways with n number, the average training time should be minimized to reflect the reduced complexity on such gateway system.

$$\begin{aligned} \min \quad & \frac{1}{n} \sum_{i=1}^n t_{train,i} \\ \text{s.t.} \quad & e < \epsilon \end{aligned} \quad (28)$$

4.2 Experimental Results

To achieve Objective 1 (i.e. improved overall accuracy), we use distributed neural-network (DNN) and soft-voting as described in section 3.3.2. To achieve Objective 2 (i.e. reduced training time), we use the same Cholesky decomposition as described in section 3.3.1

4.2.1 Setup and Benchmarks

In this section we evaluate the NSL-KDD [41] and ISCX 2012 benchmarks [42] for intrusion detection. The experiments were simulated on Ubuntu 14.04 LTS system with core 3.2GHz and 8GB RAM.

Table 5: NSL-KDD Experimental set-up parameters

Parameter	Value
Training Data Size	74258
Testing Data Size	74259
Data Dimension	41
Number of Labels	Binary (2)
	MultiClass (5)
Number of Gateways	5

Table 6: ISCX 2012 Experimental set-up parameters

Parameter	Value
Training Data Size	103285
Testing Data Size	103229
Data Dimension	11
Number of Labels	Binary (2)
	MultiClass (5)
Number of Gateways	5

4.2.2 NSL-KDD Dataset Description and Preprocessing

All the attack types mentioned previously i.e., DOS, Probe, R2L and U2R are encapsulated in the KDD Cup 99 Dataset which has been used as benchmark for detecting intrusions in a typical computer network. To evaluate the classification accuracy of SVM and SLFN as well as to evaluate detection latency using DNN we propose to use an improved version of KDD Cup 99 Dataset known as NSL-KDD dataset [41] which has been used as a benchmark in previous works on intrusion detection [21]. Some of features in the dataset i.e., protocol type, service and flag have symbolic representation. To be able to use SVM or SLFN we assigned an arbitrary sequential integer assignment to establish a correspondence between each category of a symbolic feature and a sequence of integer value. Table 5 gives the description of NSL-KDD benchmark for intrusion detection.

4.2.3 ISCX 2012 Dataset Description and Preprocessing

ISCX 2012 Dataset [42] was developed at the University of Brunswick ISCX. The original dataset contains 17 features and a label representing normal instances and intrusions belonging to DOS, SSH, L2L and Botnet (DDOS) intrusions. Some of the features were irrelevant and were removed from the dataset. Additionally some of the features in the data set i.e., appName, direction, sourceTCPFlagsDescription, destinationTCPFlagsDescription and protocolName were symbolic in nature and an arbitrary sequential integer assignment was used to convert these features to numeric features similar to NSL-KDD benchmark. Table 6 gives the description of ISCX benchmark for intrusion detection.

Table 7: Binary classification performance with 500 hidden neurons

Benchmark	Algo.	Class	TP. %	FP. %	Prec.	Recall %	FM. %	MCC	Tr.(s)	Te. (s)
NSL-KDD	SVM	normal	95.90	6.00	94.5	95.90	95.20	0.899	8678	188.7
		anomaly	94.00	4.10	95.50	94.00	94.80	0.899		
	SLFN	normal	98.24	3.77	96.54	98.24	97.38	0.945	333.08	41.94
		anomaly	96.22	1.76	98.08	96.22	97.14	0.945		
ISCX 2012	SVM	normal	98.90	1.50	99.30	98.90	99.10	0.971	5020	83.7
		anomaly	98.50	1.10	97.70	98.50	98.10	0.971		
	SLFN	normal	94.43	11.83	94.33	94.43	94.38	0.826	277.9	21.25
		anomaly	88.16	5.56	88.36	88.16	88.26	0.827		

4.2.4 Training and Testing Time Analysis

Table 7 gives the metrics for training and testing time for the two benchmarks. For NSL-KDD dataset it can be observed that SLFN is $49\times$ faster in training time compared to SVM and $8\times$ faster in testing time compared to SVM. Similarly for ISCX dataset it can be observed that SLFN is $60\times$ faster in training time compared to SVM and $13\times$ faster in testing time compared to SVM.

4.2.5 Binary and MultiClass Classification Performance Metrics

Table 7 gives the detailed performance metrics for NSL-KDD and ISCX datasets using SVM and SLFN machine learning methods for normal and anomaly classes. For NSL-KDD dataset it can be observed that all performance metrics for both normal and anomaly classes are superior for SLFN compared to SVM. For ISCX dataset SVM performs slightly better than SLFN in performance metrics for both normal and anomaly classes. However SLFN has a much higher FP rate compared to SVM.

Fig. 16: ISCX 2012 Classification

Fig. 17: NSL-KDD Classification

Fig. 16 and 17 shows the performance of SVM and SLFN using 500 hidden neurons for multiclass classification for the 2 benchmarks using the F-Measure. For ISCX dataset it can be seen that SLFN has performance comparable to SVM for Normal and Botnet classes. However SVM outperforms SLFN for DOS, SSH

Table 8: Binary DNN Classification Performance Metrics

Benchmark	Algo.	TP Rate %	FP Rate %	Prec.	Recall %	FM. %	MCC	Server (s)
NSL-KDD	SVM	95.00	5.10	95.00	95.00	95.00	0.899	188.73
	C-NIDS	95.56	4.62	95.68	95.56	95.55	0.912	41.94
	DSN(2)	95.70	4.34	95.18	95.70	95.68	0.855	12.73
	DSN(3)	95.87	4.69	95.92	95.87	95.83	0.808	8.52
	DSN(4)	95.45	4.82	95.83	95.99	95.93	0.859	6.41
	DSN(5)	95.36	4.42	95.13	95.36	95.29	0.816	5.11
ISCX 2012	SVM	98.70	1.40	98.70	98.70	98.70	0.971	83.68
	C-NIDS	91.67	10.94	91.65	91.67	91.66	0.809	20.94
	DSN(2)	91.77	10.86	91.75	91.77	91.76	0.812	10.69
	DSN(3)	91.70	11.06	91.67	91.70	91.68	0.810	7.28
	DSN(4)	91.82	11.00	91.79	91.82	91.80	0.812	5.29
	DSN(5)	91.62	10.99	91.60	91.62	91.61	0.808	4.28

and L2L classes. For NSL-KDD dataset it can be observed that for Normal, DOS and Probe classes SLFN has almost similar performance compared to SVM. SLFN outperforms SVM for R2L class. However it is not able to detect any intrusions relating to U2R class. This can be attributed to the fact that the NSL-KDD dataset only contains 119 instances of U2R class.

4.2.6 DNN Performance

Table 8 shows the performance metrics for DNN using NSL-KDD and ISCX dataset-
s at 500 hidden neurons. It can be observed that for both datasets all the performance metrics stay relatively constant however the server processing time is reduced proportionally to the number of sub neural networks employed. For NSL-KDD server processing time is reduced by $8\times$ compared to centralized NIDS and by $37\times$ compared to SVM when using 5 sub neural networks. Similarly for ISCX dataset server processing time is reduced by $5\times$ compared to centralized NIDS and by $20\times$ when compared to SVM when using 5 sub neural networks. The reduced server processing time allows DNN to detect intrusions in lesser amount of time since each SLFN in DNN has a reduced detection latency.

5 Conclusion

In this chapter, we have discussed the application of computational intelligence techniques for indoor data analytics on smart-gateway network. Firstly, a computational efficient data analytic platform is introduced for smart home management system based on the distributed gateway network. Secondly, a distributed support vector machine (DSVM) and distributed neural network (DNN) based machine learning algorithm are introduced for the indoor data analytics, which can significantly reduce the training complexity and training time yet maintaining acceptable accuracy. We have applied the DSVM and DNN for indoor positioning to analyze the WiFi data; and further to analyze network intrusion detection to provide the network security. Such a computational intelligence technique can be compactly realized on the computational-resource limited smart-gateway networks, which is

desirable to build a real cyber-physical system towards future smart home, smart building, smart community and further a smart city.

Acknowledgement: this work is sponsored by grants from Singapore MOE Tier-2 (MOE2015-T2-2-013), NRF-ENIC-SERTD-SMES-NTUJTC13C-2016 (WP4) and NRF-ENIC-SERTD-SMES-NTUJTC13C-2016 (WP5).

References

1. J. Kacprzyk and W. Pedrycz, *Springer handbook of computational intelligence*. Springer, 2015.
2. D. L. Poole, A. K. Mackworth, and R. Goebel, *Computational intelligence: a logical approach*. Oxford University Press New York, 1998, vol. 1.
3. H.-K. Lam and H. T. Nguyen, *Computational intelligence and its applications: evolutionary computation, fuzzy logic, neural network and support vector machine techniques*. World Scientific, 2012.
4. D. Peleg, "Distributed computing," *SIAM Monographs on discrete mathematics and applications*, vol. 5, 2000.
5. W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 190–203, 2016.
6. Apache hadoop. [Online]. Available: <http://hadoop.apache.org/>
7. K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, H. Cui, and E. Y. Chang, "Parallelizing support vector machines on distributed computers," in *Advances in Neural Information Processing Systems*, 2008, pp. 257–264.
8. B. Barker, "Message passing interface (MPI)," in *Workshop: High Performance Computing on Stampede*, 2015.
9. H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 6, pp. 1067–1080, 2007.
10. C. Yang and H.-R. Shao, "Wifi-based indoor positioning," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 150–157, 2015.
11. A. M. Hossain and W.-S. Soh, "A survey of calibration-free indoor positioning systems," *Computer Communications*, vol. 66, pp. 1–13, 2015.
12. S. He and S.-H. G. Chan, "Wi-fi fingerprint-based indoor positioning: Recent advances and comparisons," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 466–490, 2016.
13. J. Xu, W. Liu, F. Lang, Y. Zhang, and C. Wang, "Distance measurement model based on rssi in wsn," *Wireless Sensor Network*, vol. 2, no. 08, p. 606, 2010.
14. J. Torres-Solis, T. H. Falk, and T. Chau, *A review of indoor localization technologies: towards navigational assistance for topographical disorientation*. INTECH Publisher, 2010.
15. J. Janicka and J. Rapinski, "Application of rssi based navigation in indoor positioning," in *Geodetic Congress (Geomatics), Baltic*. IEEE, 2016, pp. 45–50.
16. Y. Cai, S. K. Rai, and H. Yu, "Indoor positioning by distributed machine-learning based data analytics on smart gateway network," in *Indoor Positioning and Indoor Navigation (IPIN), 2015 International Conference on*. IEEE, 2015, pp. 1–8.
17. J. P. Anderson, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.
18. E. Nyakundi, "Using support vector machines in anomaly intrusion detection," Ph.D. dissertation, The University of Guelph, 2015.
19. D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, "A survey of distance and similarity measures used within network intrusion anomaly detection," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 70–91, 2015.
20. C.-C. Sun, C.-C. Liu, and J. Xie, "Cyber-physical system security of a power grid: State-of-the-art," *Electronics*, vol. 5, no. 3, p. 40, 2016.

21. Y. Zhang, L. Wang, W. Sun, R. C. Green II, and M. Alam, "Distributed intrusion detection system in a multi-layer network architecture of smart grids," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 796–808, 2011.
22. D. J. Cook, J. C. Augusto, and V. R. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," *Pervasive and Mobile Computing*, vol. 5, no. 4, pp. 277–298, 2009.
23. S. Nikolettseas, M. Rapti, T. P. Raptis, and K. Veroutis, "Decentralizing and adding portability to an iot test-bed through smartphones," in *IEEE DCOSS*, 2014.
24. C. Zhang, W. Wu, H. Huang, and H. Yu, "Fair energy resource allocation by minority game algorithm for smart buildings," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 63–68.
25. H. Huang, Y. Cai, H. Xu, and H. Yu, "A multi-agent minority-game based demand-response management of smart buildings towards peak load reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2016.
26. S. Helal, B. Winkler, C. Lee, Y. Kaddoura, L. Ran, C. Giraldo, S. Kuchibhotla, and W. Mann, "Enabling location-aware pervasive computing applications for the elderly," in *IEEE PerCom*, 2003.
27. C. Drane, M. Macnaughtan, and C. Scott, "Positioning GSM telephones," *Communications Magazine, IEEE*, vol. 36, no. 4, pp. 46–54, 1998.
28. M. O. Ergin, V. Handziski, and A. Wolisz, "Node sequence discovery in wireless sensor networks," in *IEEE DCOSS*, 2013.
29. C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.
30. C. Zhang, P. Li, A. Rajendran, and Y. Deng, "Parallel multicategory support vector machines (pmc-svm) for classifying microarray data," in *Computer and Computational Sciences, 2006. IMSCC-S'06. First International Multi-Symposiums on*, vol. 1. IEEE, 2006, pp. 110–115.
31. C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.
32. G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
33. D. L. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.
34. H. Huang, Y. Cai, and H. Yu, "Distributed-neuron-network based machine learning on smart-gateway network towards real-time indoor data analytics," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 720–725.
35. G. Feng, G.-B. Huang, Q. Lin, and R. Gay, "Error minimized extreme learning machine with growth of hidden nodes and incremental learning," *Neural Networks, IEEE Transactions on*, vol. 20, no. 8, pp. 1352–1357, 2009.
36. L. N. Trefethen and D. Bau III, *Numerical linear algebra*. Siam, 1997, vol. 50.
37. D. J. Miller and L. Yan, "Critic-driven ensemble classification," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2833–2844, 1999.
38. P. Bahl and V. N. Padmanabhan, "Radar: An in-building rf-based user location and tracking system," in *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, 2000.
39. M. Brunato and R. Battiti, "Statistical learning theory for location fingerprinting in wireless LANs," *Computer Networks*, vol. 47, no. 6, pp. 825–845, 2005.
40. I. Ekahau. (2015). [Online]. Available: <http://www.test.org/doi/>
41. NSL-KDD dataset. [Online]. Available: <http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html>
42. ISCX 2012 dataset. [Online]. Available: <http://www.unb.ca/research/iscx/dataset/iscx-IDS-dataset.html>