

Metadata of the chapter that will be visualized in SpringerLink

Book Title	Convergence of Artificial Intelligence and the Internet of Things	
Series Title		
Chapter Title	In-Network Machine Learning Predictive Analytics: A Swarm Intelligence Approach	
Copyright Year	2020	
Copyright HolderName	Springer Nature Switzerland AG	
Corresponding Author	Family Name	Ivanov
	Particle	
	Given Name	Hristo
	Prefix	
	Suffix	
	Role	
	Division	Essence: Pervasive & Distributed Intelligence Research Group
	Organization	School of Computing Science, University of Glasgow
	Address	G12 8QQ, Glasgow, UK
	Email	2205747i@student.gla.ac.uk
Author	Family Name	Anagnostopoulos
	Particle	
	Given Name	Christos
	Prefix	
	Suffix	
	Role	
	Division	Essence: Pervasive & Distributed Intelligence Research Group
	Organization	School of Computing Science, University of Glasgow
	Address	G12 8QQ, Glasgow, UK
	Email	christos.anagnostopoulos@glasgow.ac.uk
Author	Family Name	Kolomvatsos
	Particle	
	Given Name	Kostas
	Prefix	
	Suffix	
	Role	
	Division	Essence: Pervasive & Distributed Intelligence Research Group
	Organization	School of Computing Science, University of Glasgow
	Address	G12 8QQ, Glasgow, UK
	Email	kostas.kolomvatsos@glasgow.ac.uk
Abstract	<p>This chapter addresses the problem of collaborative Predictive Modelling via in-network processing of contextual information captured in Internet of Things (IoT) environments. In-network predictive modelling allows the computing and sensing devices to disseminate only their local predictive Machine Learning (ML) models instead of their local contextual data. The data center, which can be an Edge Gate- way or the Cloud, aggregates these local ML predictive models to predict future outcomes. Given that communication between devices in IoT environments and a centralised data center is energy consuming and</p>	

communication bandwidth demanding, the local ML predictive models in our proposed in-network processing are trained using Swarm Intelligence for disseminating only their parameters within the network. We further investigate whether dissemination overhead of local ML predictive models can be reduced by sending only relevant ML models to the data center. This is achieved since each IoT node adopts the Particle Swarm Optimisation algorithm to locally train ML models and then collaboratively with their network neighbours one representative IoT node fuses the local ML models. We provide comprehensive experiments over Random and Small World network models using linear and non-linear regression ML models to demonstrate the impact on the predictive accuracy and the benefit of communication-aware in-network predictive modelling in IoT environments.

In-Network Machine Learning Predictive Analytics: A Swarm Intelligence Approach



Hristo Ivanov, Christos Anagnostopoulos, and Kostas Kolomvatsos

Abstract This chapter addresses the problem of collaborative Predictive Modelling via in-network processing of contextual information captured in Internet of Things (IoT) environments. In-network predictive modelling allows the computing and sensing devices to disseminate only their local predictive Machine Learning (ML) models instead of their local contextual data. The data center, which can be an Edge Gate- way or the Cloud, aggregates these local ML predictive models to predict future outcomes. Given that communication between devices in IoT environments and a centralised data center is energy consuming and communication bandwidth demanding, the local ML predictive models in our proposed in-network processing are trained using Swarm Intelligence for disseminating only their parameters within the network. We further investigate whether dissemination overhead of local ML predictive models can be reduced by sending only relevant ML models to the data center. This is achieved since each IoT node adopts the Particle Swarm Optimisation algorithm to locally train ML models and then collaboratively with their network neighbours one representative IoT node fuses the local ML models. We provide comprehensive experiments over Random and Small World network models using linear and non-linear regression ML models to demonstrate the impact on the predictive accuracy and the benefit of communication-aware in-network predictive modelling in IoT environments.

H. Ivanov (✉) · C. Anagnostopoulos · K. Kolomvatsos

Essence: Pervasive & Distributed Intelligence Research Group, School of Computing Science, University of Glasgow, G12 8QQ Glasgow, UK
e-mail: 2205747i@student.gla.ac.uk

C. Anagnostopoulos

e-mail: christos.anagnostopoulos@glasgow.ac.uk

K. Kolomvatsos

e-mail: kostas.kolomvatsos@glasgow.ac.uk

© Springer Nature Switzerland AG 2020

G. Mastorakis et al. (eds.), *Convergence of Artificial Intelligence and the Internet of Things*, Internet of Things, https://doi.org/10.1007/978-3-030-44907-0_7



1 Introduction

This section introduces the aims of the chapter, the motivations behind it and the structure of this chapter.

1.1 Motivation

Internet of Things is defined as a “proposed development of the Internet in which everyday objects have network connectivity, allowing them to send and receive data [21].” Every IoT device has an IP address and is able to transfer data over a network. IoT devices become more popular with each passing year. In 2018 their number hit 7 billion and there are no signs of them slowing down [18].

Nowadays, IoT devices have numerous applications and almost every industry uses some sort of smart devices to automate processes, gather data or communicate with other devices. On the other hand, there are many challenges that they face. Some of them are security, connectivity, compatibility and privacy, but the main talking point of this chapter would be energy consumption and how we can reduce it (Fig. 1).

This chapter focuses on one use case of Internet of Things devices **Wireless Sensor Networks (WSN)**. In this case, energy conservation is everything. The longer a sensor can be kept alive, the more data you can gather and less often you will have to recharge the device (which may be difficult, depending on the conditions sensors are located at). One way to address that problem is **Predictive Modelling**. Predictive Modelling is a process that uses statistics and machine learning models to predict outcomes (closely related to Predictive Analytics). It can never predict the future, but it can look at existing data and determine a likely outcome [11]. The benefits of predictive

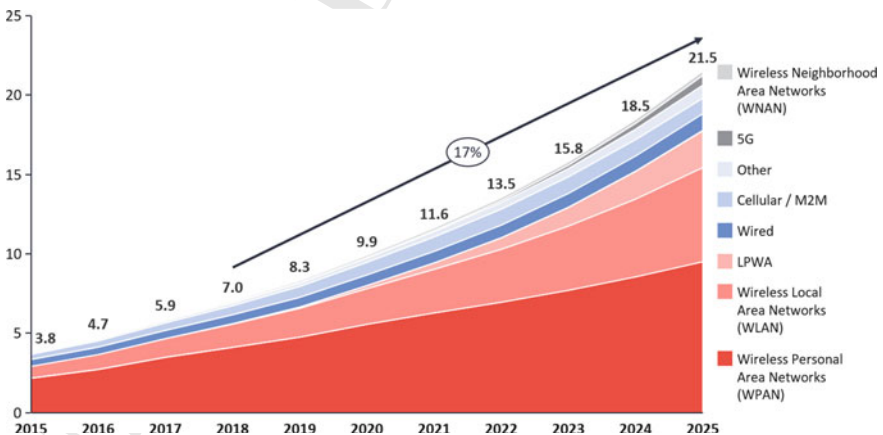


Fig. 1 Number of global IoT Connections in Billions [18]

modelling are that it brings the computation to the data and not vice versa. Sensing and computing devices do not have to send their entire local data over the network to the **Data Center** (or **sink node**), but just have to generate a local machine learning model trained on that data and send that instead. In the long term, this approach should greatly benefit the battery life of the sensing and computing devices. Moreover, it would allow the data center (DC) to study the received machine learning models, predict possible outcomes of that data and recreate part of the datasets. Of course, there are some drawbacks to this approach—adaptability of the ML models to new data (the models will not always be up to date) and models cannot be 100% accurate (if accuracy is critical this might not be a suitable approach).

1.2 Aim

The objective of this chapter is to investigate whether we could reduce the communication overhead of the sensing and computing devices (nodes) by implementing different network models and reducing the number of nodes communicating with the back-end data center system, e.g., sink node. Each device has a unique dataset and will use the **Particle Swarm Optimisation** (Swarm Intelligence) algorithm to calculate the regression coefficients (linear or non-linear) that best fit its data.

This chapter will implement and analyse two network types—Random network and Small World network. These network models will be used to generate node neighbourhoods. For each neighbourhood, we will generate a fused regression model which will ‘generalise’ all the local regression models in that group. One representative of each group will be responsible for collecting the local ML models of its neighbours and using the proposed methodology of this chapter to generate that fused model. After that, each of those models will be sent to the sink node, to evaluate their accuracy using a test dataset. Furthermore, the proposed methodology’s performance will be evaluated against a baseline solution, in terms of energy consumption and prediction accuracy of each machine learning model. This communication overhead—prediction accuracy tradeoff will be an important talking point in the chapter.

1.3 Outline

This section introduced the motivations and aims behind the chapter. The remainder of this chapter is structured as follows:

- **Section 2—Background** introduces the reader to topics that are relevant to this chapter and reviews relevant literature.
- **Section 3—Analysis** defines the problem this chapter is trying to solve, outlines the baseline solution, proposed methodology and network models in greater detail.

- **Section 4—Implementation** describes the approach taken to solve the challenges mentioned in the Analysis section and outlines the implementation details of this chapter's baseline solution, proposed methodology and network models.
- **Section 5—Performance and Comparative Assessment** evaluates the performance of the proposed methodology against the baseline solution and discusses the tradeoff between communication overhead and prediction accuracy of the regression models.
- **Section 6—Conclusion** provides reflections on the proposed methodology, draws conclusions on its effectiveness and performance, identifies opportunities for future work and summarises the whole chapter.

2 Background

This section summarises several books, research papers and topics that I had to get familiar with and algorithms that had to be implemented as part of the chapter.

2.1 Particle Swarm Optimisation Algorithm

The Particle Swarm Optimisation (PSO) algorithm is a population-based search and optimisation algorithm based on the simulation of the social behaviour of birds within a flock [9]. It was developed by Dr. Eberhart and Dr. Kennedy in 1995. The PSO algorithm has many advantages including simplicity of implementation, reliability, robustness, and in general, is considered an effective meta-heuristic optimisation algorithm [23]. Kennedy and Eberhart [17] describe the algorithm as a very simple concept, for which paradigms can be implemented in a few lines of code. It requires only primitive mathematical operators and is computationally inexpensive in terms of both memory requirements and speed.

The algorithm works by having a population of candidate solutions (particles) and moving them around in the search-space according to simple mathematical formulae over their position and velocity. Each particle's movement is influenced by its local best known position but is also guided toward the best known position in the search space, which is updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions [33] (see plot 2).

The PSO algorithm is used in many fields because optimisation is found everywhere. In the production of a new device, in a new artificial intelligence technique, in a big data application or in a deep learning network, optimisation is the most important part of an application. The PSO algorithm is used in the chapter to find the optimal coefficients in the regression model equations (see 4 or 2.5).

Özsoy and Örkücü [23] used the PSO algorithm in their paper to estimate non-linear regression model parameters. They note that the algorithm exhibits a rapid

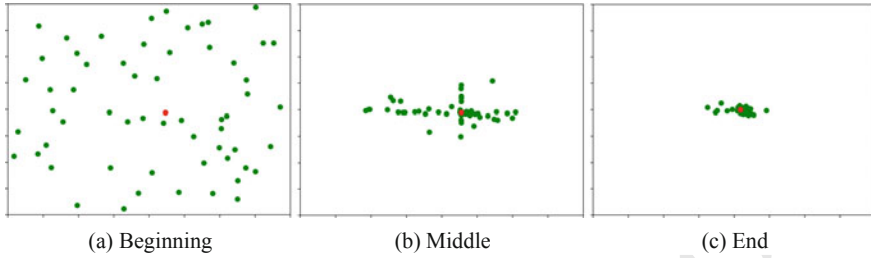


Fig. 2 Particle positions at different stages of the PSO algorithm

convergence tendency and it specifically converged after at most 20–25 iterations for all the models with the number of parameters ranging from 2 to 9. They conclude that the algorithm is a very efficient method for handling the problems of parameter estimation of nonlinear regression models (Fig. 2).

2.2 Particle Representation

Kennedy and Eberhart converted the social simulation algorithm into an optimisation paradigm. The idea was to let the agents (birds) find an unknown favourable place in the search space (food source) through capitalising on one another's knowledge. Each agent is able to remember its best position and knows the best position of the whole swarm. The extremum of the mathematical function to be optimised can be thought of as the food source [15].

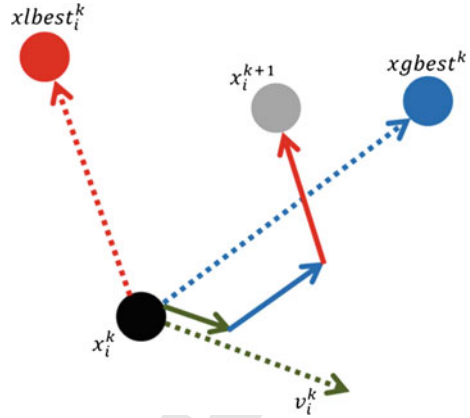
In the algorithm each bird is represented by a particle. In this chapter each particle represents a possible estimation of coefficients (e.g. $p = [w0, w1]$). Particles move around the search coefficient space (by adding a calculated velocity to their position at every iteration) to find the optimal coefficients $[w0^*, w1^*]$. The following two equations calculate the velocity and the position of a particle, respectively:

$$v_{i,j}^{k+1} = wv_{i,j}^k + c_1r_1(xbest_{i,j}^k - x_{i,j}^k) + c_2r_2(gbest_j^k - x_{i,j}^k) \quad (1)$$

$$x_{i,j}^{k+1} = x_{i,j}^k + v_{i,j}^{k+1}, \quad (2)$$

where $x_{i,j}$ and $v_{i,j}$ are the position and velocity of the j component of the x particle at iteration i ; $gbest_j$ and $xbest_{i,j}$ show the best position achieved so far by the whole swarm in dimension j and the personal best for particle i , respectively; r_1 and r_2 are random numbers in the interval (0,1) sampled from a uniform distribution; c_1, c_2 are the cognition factor (particle's confidence in itself) and the social factor (particle's confidence in the swarm) [15]; w is the inertia weight (weighs the contribution of the velocity from the previous iteration).

Fig. 3 The movement of a single particle based on Eqs. 1, 2



Particle positions need to be randomly initialised in order to cover as much search space as possible and to distribute them so that there is a greater chance the optimum is found. The initialisation of particles plays an important role in the performance of the algorithm. If it is not appropriate then the particles may end up searching in the wrong area, get stuck in a local minimum or find a suboptimal solution. A good technique that is used in this chapter is the following: if each dimension j is defined by the two vectors, $x_{min,j}$ and $x_{max,j}$, which respectively represent the minimum and maximum positions, then the particle's position is initialised to: (Fig. 3)

$$x = x_{min,j} + r_j(x_{max,j} - x_{min,j}), \forall j = 1, \dots, n, \quad (3)$$

where $r_j \sim U(0, 1)$ and n is the number of dimensions [9].

2.3 Parameters

The performance of PSO depends greatly on its parameters. The optimal values for these parameters are dependent on the problem itself, the search space and the convergence time required. The control parameters of the algorithm are namely the dimension of the problem, number of particles, inertia weight, number of iterations and the cognitive and social components. Additionally, if velocity clamping (setting a limit on particles' velocity) or constriction (an approach similar to adding inertia weight) is used, the maximum velocity and constriction coefficient also influence the performance of the PSO [9]. This chapter has experimented with different values for these parameters which will be discussed in detail in Sect. 4—Implementation.

2.4 The PSO Algorithm

Each position in the n -dimensional search space corresponds to the values of n regression coefficients. During each iteration of the algorithm, each solution is evaluated by an objective function to determine its fitness. All particles will be moving through the search space to find the position for which the fitness value is best (minimum or maximum, depending on the problem). Algorithm 1 shows the pseudo code for PSO. The Particle Swarm Optimisation algorithm will be discussed in greater detail later in this work—Sect. 4—Implementation where it will be used to generate regression coefficients for the local machine learning models.

Algorithm 1: Pseudo code for the PSO algorithm.

```

Data:  $P$ , an array of particles
 $N$  the number of particles
 $D$  the number of dimensions
 $F$  the fitness function
 $g_{best}$  the global best
begin

     $g_{best} \leftarrow MAXINT$ 
    for  $i < N$  do

        for  $j < D$  do

            Initialise  $P_{i,j}$ .position
            Initialise  $P_{i,j}$ .velocity

        end

    end

    for  $i < N$  do

         $P[i].best = F(P[i].position)$ 
        if  $P[i].best < g_{best}$  then
             $g_{best} = P[i].best$ 
        end

    end

    while max iteration not reached or termination condition not met do

        for  $i < N$  do

            if  $F(P[i].position) < P[i].best$  then

                 $P[i].best = F(P[i].position)$ 
                if  $P[i].best < g_{best}$  then

                     $g_{best} = P[i].best$ 

```

```

197  end
198  end
199  end
200  for  $i < N$  do
201      for  $j < D$  do
202          Update  $P_{i,j}$ .velocity
203           $P_{i,j}$ .position =  $P_{i,j}$ .position +  $P_{i,j}$ .velocity
204      end
205  end
206  end
207  end

```

2.5 Regression

Regression is a statistical measurement that attempts to determine the strength of the relationship between one dependent variable (usually denoted by y) and a series of other changing variables (known as independent variables, denoted by x) [5]. Regression plays an important part in this chapter, as each device would have to compute its local regression coefficients.

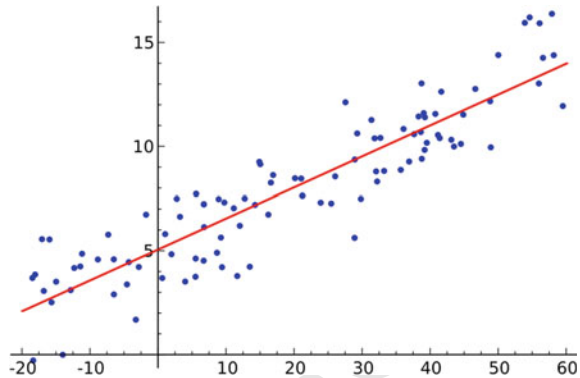
Linear Regression Model

There are two types of linear regression: simple and multiple. Simple linear regression is useful for finding a relationship between two continuous variables. One is a predictor (independent variable) and the other is a response (dependent variable). It looks for a statistical relationship but not deterministic relationship [28]. In the simple linear regression model, there are two parameters—slope and intercept. The slope is the ratio of the change in the y -value over the change in the x -value and the intercept is the y -value of the point where the line intersects the y -axis [20]. The general form of a simple linear regression model looks like this:

$$y = w_0 * x + w_1, \quad (4)$$

where w_0 is the slope, w_1 —the intercept, x is the independent variable and y —dependent variable. Linear regression is a linear approach to model the relationship between a scalar response and one or more explanatory variables. For more than one explanatory variable, the process is called multiple linear regression [34] (Fig. 4).

The core idea of linear regression is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (of all data points) is as small as possible. The error is the distance between the point and the regression line [28]. Sometimes the relationship between variables is not linear and needs a

Fig. 4 Linear regression line

more complicated nonlinear equation, that is why this chapter also experiments with nonlinear regression models.

Nonlinear Regression Model

While a linear regression has one form, nonlinear regression can take many forms. If the equation does not satisfy the requirements for linearity, then it is not linear. Nonlinear models are used to model complex interrelationships among variables and play an important role in various scientific disciplines and engineering [23]. A nonlinear regression equation looks like this:

$$y = f(\beta, x) + \varepsilon, \quad (5)$$

where x is a vector of p predictors, β is a vector of k parameters, f is some known regression function, and ε is an error term whose distribution may or may not be normal [31]. Nonlinear regression is characterised by the fact that the prediction equation depends nonlinearly on one or more unknown parameters [23]. Here are just a few nonlinear regression models that have been used in the implementation of this chapter—Logistic, Jennrich and Meyer 1 models respectively:

$$\frac{\beta_1}{1 + \exp(\beta_2 - \beta_3 x)} \quad (6)$$

$$\exp(\beta_1 x) + \exp(\beta_2 x) \quad (7)$$

$$\frac{\beta_1 \beta_3 x_1}{1 + \beta_1 x_1 + \beta_2 x_2} \quad (8)$$

Nonlinear models make the estimation of parameters and the statistical analysis of parameter estimates more difficult and challenging. Difficulties arise due to the large number of parameters and the multimodal nature of the objective function. It is very difficult to minimise a sum of squared errors function using ordinary optimisation techniques. In order to overcome those difficulties, the use of a powerful

meta-heuristic method such as the Particle Swarm Optimisation (PSO) algorithm is needed [23].

2.6 Prediction Error Metrics

Predictive modelling works on the constructive feedback principle. You build a model, get feedback from metrics, make improvements and continue until you achieve a desirable accuracy. Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results [27]. Since our models will produce an output given any input or set of inputs, we can then check these estimated outputs against the actual values that we tried to predict. The difference between the actual value and the model's estimate is called a residual. The residual can be calculated for every point in the data set, and each of these residuals will be of use in the assessment. These residuals will play a significant role in judging the usefulness of a model. If the residuals are small, it implies that the model that produced them does a good job of predicting the output of interest. Conversely, if these residuals are generally large, it implies that the model is a poor estimator [24].

This chapter will use the **Root Mean Square Error (RMSE)** as an error metric to evaluate all regression models. The RMSE is the distance, on average, of a data point from the fitted line, measured along a vertical line (see Eq. 9). RMSE is directly interpretable in terms of measurement units, and so is a better measure of goodness of fit than a correlation coefficient [29]. The RMSE is calculated as follows:

$$F = \sqrt{\sum_{i=1}^n (y_i - \hat{y}_i)^2}, \quad (9)$$

where y_i is actual y in the dataset and \hat{y}_i is the predicted y (Fig. 5).

Moreover, RMSE has the benefit of penalising large errors and avoids the use of taking the absolute value, because of those reasons it is more appropriate than other error metrics, e.g. MAE (Mean Absolute Error) [13].

2.7 Network Modelling

Random Network

A random network (or random graph) is a graph where edges are created by a random process. It is obtained by starting with a set of n isolated vertices and adding successive edges between them at random. Different random graph models produce

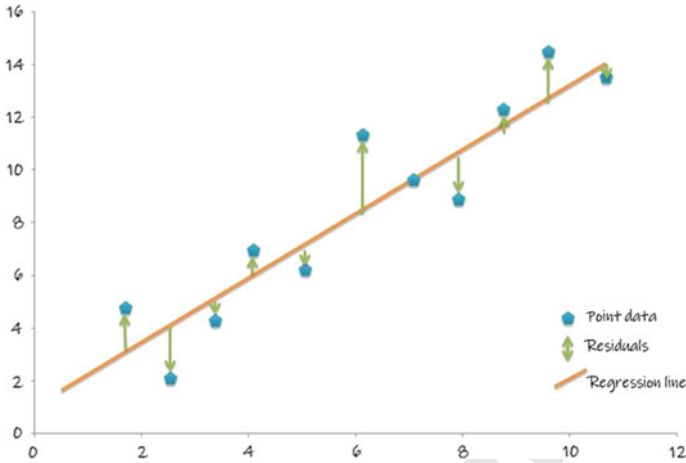


Fig. 5 Visualisation of the Root Mean Square Error [30]

different probability distributions on graphs. Most commonly studied is the one proposed by Edgar Gilbert, which is used in this chapter, denoted $G(n, p)$, in which every possible edge occurs independently with probability $0 < p < 1$. The probability of obtaining any one particular random graph with m edges is:

$$p^m (1 - p)^{N-m}, N = \binom{n}{2} [35]. \quad (10)$$

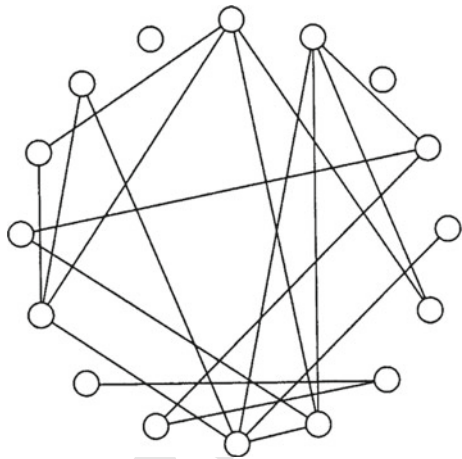
Therefore, by increasing the probability p the network becomes denser and the size of the node neighbourhoods increases. By having large neighbourhoods, we expect to have increased communication costs (node is communicating with more neighbours) and increased accuracy (the model of the neighbourhood is trained on more data). In this type of networks the difference between node degrees is not big and therefore there are rarely outliers, which makes it suitable for this chapter [4] (Fig. 6).

The connectivity links in the random network implementation used in this chapter are unidirectional. This is done because the goal is to have different neighbourhoods per node, in which neighbourhood that node receives the models from its neighbours and generates the fused model of the group.

Small World Network

The small world phenomenon, also known as six degrees of separation states that if you choose any two individuals anywhere on Earth, you will find a path of at most six acquaintances between them [4]. The Watts and Strogatz model small world networks are used in this chapter. This model is a random graph generation model that produces graphs with small-world properties, including short average path lengths and high

Fig. 6 An example of a standard random network [8]



clustering. It was proposed by Duncan J. Watts and Steven Strogatz in their joint 1998 Nature paper [32].

The algorithm for creating a small world network is the following:

- Create a ring structure from the n nodes.
- Connect every node to its k closest neighbours ($k = 1$ when k is odd).
- For every node n_i take every edge n_i, n_j and rewire it with probability p .
- Rewiring is done by removing the edge n_i, n_j and replacing it with n_i, n_k , where k is chosen at random.

p is the rewiring probability of the network. As we can see from the plot above (Fig. 7) when p is equal to 0, there are no rewirings and each node is connected to its k closest neighbours. By increasing the probability, we increase the chance that a certain node will be disconnected with one of its neighbours and become connected to another random node. The mean number of neighbours per node— k stays the same, regardless of the value of p . As the value of p increases, the network starts behaving like a random network with a constant mean number of neighbours per node.

2.8 Mica2 Wireless Sensor Platform

The energy consumption model from the Mica2 sensor board was adopted in this chapter. Mica2 is a wireless platform and serves as a foundation for various wireless server network applications. We will take into account the energy costs for instruction execution, transmitting and receiving bits when we evaluate the energy consumption of regression models in Sect. 5—Performance and Comparative Assessment.

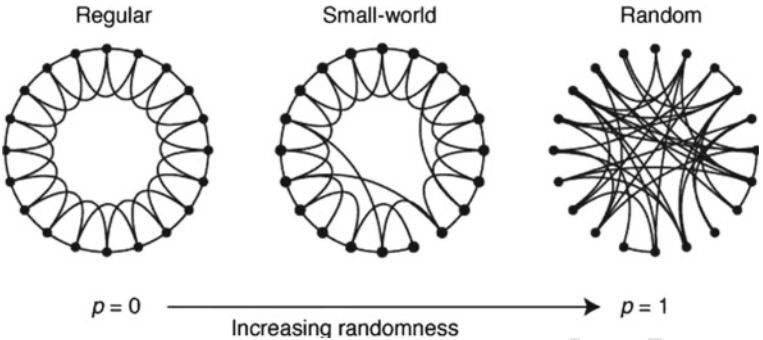


Fig. 7 Watts Strogatz small world networks with different rewiring probabilities p

Table 1 Energy costs of some Mica2 operations

Node operation mode	Energy cost
Instruction execution	4 nJ/instruction
Transmitting	720 nJ/bit
Receiving	110 nJ/bit

This energy model assumes the energy is coming out of two AA batteries that approximately supply 2,200 mAh with an effective average voltage of 3 V. It consumes 20 mA if running a sensing application continuously, which leads to a lifetime of 100 h [1]. The energy costs of the Mica2 operations are summarised in Table 1.

The packet header of the communication protocol adopted by Mica2 is 9 bytes (MAC header and CRC¹) and the maximum payload is 29 bytes. Therefore, the perpacket overhead equals 23.7% (lowest value). This is quite high, thus necessitating a multifield transmission solution (i.e., packing multiple contextual values in the same WSN² message) [1]. We assume the sensors will be sensing with a lot of precision. Consequently, for each contextual value, the assumed payload is 8 bytes (high precision floating point number). Therefore, we would be able to fit two float values in each packet. This will be taken into account when calculating the energy costs of each machine learning model.

3 Analysis

This section talks about the problem this chapter is trying to solve, the baseline solution and its limitations and the proposed solution.

¹Cyclic Redundancy Check.

²Wireless Sensor Network.

3.1 Problem Definition and Analysis

Advances in microelectronics technology have made it possible to build inexpensive, low-power, miniature sensing devices. Equipped with a microprocessor, memory, radio, and battery, such devices can now combine the functions of sensing, computing, and wireless communication into miniature smart sensor nodes [19].

Challenges in sensor networks include high bandwidth demand, high energy consumption, quality of service (QoS) provisioning and data processing. However, the energy constraint is unlikely to be solved soon due to slow progress in developing battery capacity. Moreover, the untended nature of sensor nodes and hazardous sensing environments preclude battery replacement as a feasible solution [12]. Predictive Modelling is an efficient way of reducing those communication costs, by generating a local machine learning model of the data within the sensing nodes and sending that model to the data center, which will be able to predict possible future outcomes of that data.

At its core, this chapter tries to solve the contradiction between service quality and survival time of IoT sensor networks. We are aiming to increase the survival time of the sensing and computing devices by significantly reducing the number of nodes communicating with the sink node (very costly) by exploiting node neighbourhoods. Only a representative of each neighbourhood will send the regression model, that is trained on all the data in the group, to the data center. This will lead to trading off communication overhead and accuracy which we will talk about in more detail in Sect. 5—Performance and Comparative Assessment. The base case network will not be generating neighbourhoods and every sensing device will communicate with the data center directly. This baseline solution will be useful for comparing the performance of the proposed methodology against it (Fig. 8).

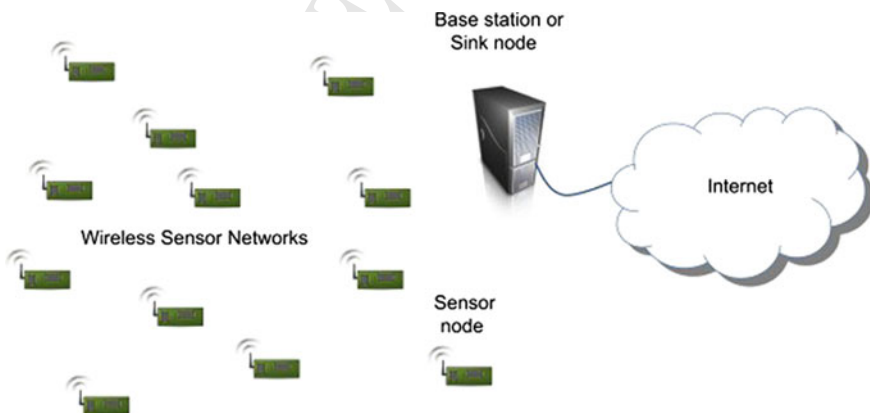


Fig. 8 A simple wireless sensor network

3.2 Baseline Methodology

In our base case solution, every node talks to the sink node. We do not have any neighbourhoods of nodes and we assume no collaboration between them. There are two ways of predictive modelling in this case: **Global Model** and **Average Model**.

Global Model

Every node n_i sends its dataset d_i (via wireless links) to the sink node. In this ideal case (very energy expensive) the sink node collects all the data and trains a machine learning model on that data and the result is the global model. This model is expected to be the most accurate because it is trained on the entire dataset (aggregation of the local datasets of all sensing devices) (Fig. 9).

Average Model

Another reference model is the Average Model. In this case, we assume that the sink node is not collecting all the data. Instead, devices are running the PSO algorithm on their own datasets in order to generate local regression coefficients— f_i . After that, each device sends its local model to the sink node. Then, the sink node is averaging all the local models to generate an aggregate model which is equal to:

$$\frac{1}{n} \sum_{i=1}^n f_i, \quad (11)$$

where n is the number of nodes in the network. This model takes the naive approach of averaging all the parameters (in the linear regression case—local slopes and local offsets) and generates the new aggregate model. E.g., if we have two local linear models: $y_1 = 3x + 4$ and $y_2 = 4x - 3$ then the average model is equal to: $y = (3x + 4x) * 0.5 + (4 - 3) * 0.5 = 3.5x + 0.5$. The average model is going to be our point of

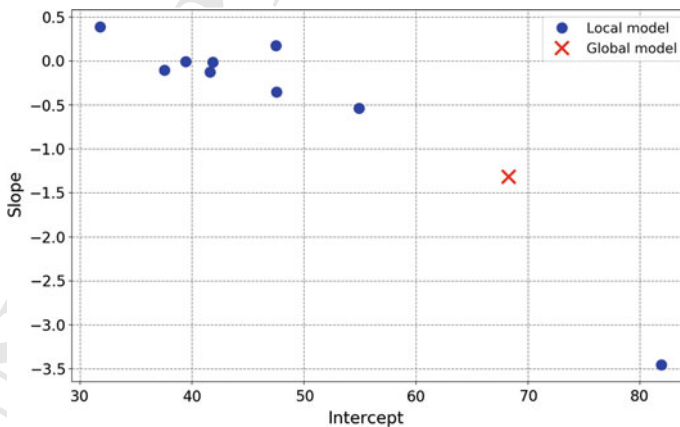


Fig. 9 Simple Linear Regression models of each node, including the Global model for comparison

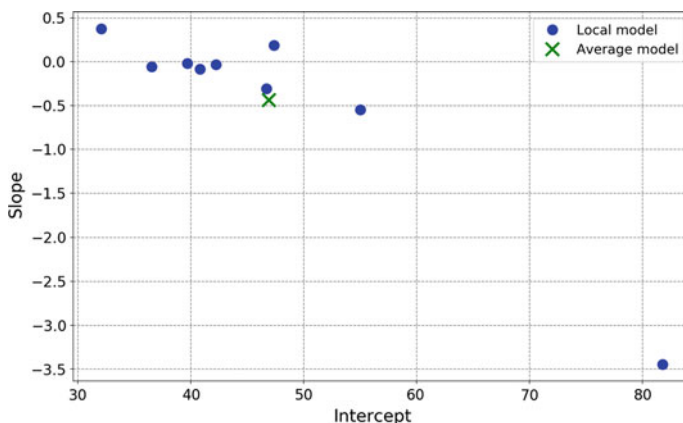


Fig. 10 Simple Linear Regression models of each node, including the Average model for comparison

reference for energy efficiency, because very low energy consumption is required to send only local predictive models per device, instead of contextual data (Fig. 10).

3.3 Limitations

There are certain limitations associated with the two reference models mentioned above. The global model could prove very costly for each sensing device because it means they have to send their entire datasets over the network, which is too expensive and very much hurts scalability. This approach will require the devices to read data from disk, load it into memory and send it over the network to the data center which overall will be very expensive in terms of energy cost (see transmission costs per bit from Table 1).

The average model introduces less communication overhead than the global model. This is because the model generation phase is performed locally at each node and only those local models are sent to the receiver. On the other hand, its accuracy is expected to be worse because of the naive approach of taking the average of all the received regression models. This method is naive because it does not take into consideration the size of the different datasets and the ‘mass’ of (x, y) pairs in the 2D plane (Fig. 10).

Overall, generating the global and average models will lead to high energy consumption and decreased prediction accuracy, respectfully. Both have their limitations and neither is the obviously better choice.

3.4 Proposed Methodology

This section lists the required steps of the proposed methodology.

Generate Neighbourhoods

The proposed methodology will exploit node networking neighbourhoods. Therefore, the first step will be to generate those neighbourhoods. Depending on the network type, initialise an adjacency matrix $adj[][]$ where $adj[i][j] == 1$ if and only if node j is part of the neighbourhood of node i . In that way, each node creates a neighbourhood of its own, which will be exploited to generate a fused regression model that fits the aggregated dataset from the nodes in the group.

The Local PSO Model

After that, every sensing device needs to individually analyse its contextual data— d_i and generate its local regression coefficients. Each node evidently derives a different regression model and in order to generate that model, it runs the Particle Swarm Optimisation algorithm and gets the optimal local regression coefficients of the model that best fits its local data— $PSO(d_i) \Rightarrow (w_0, w_1, \dots, w_n)$.

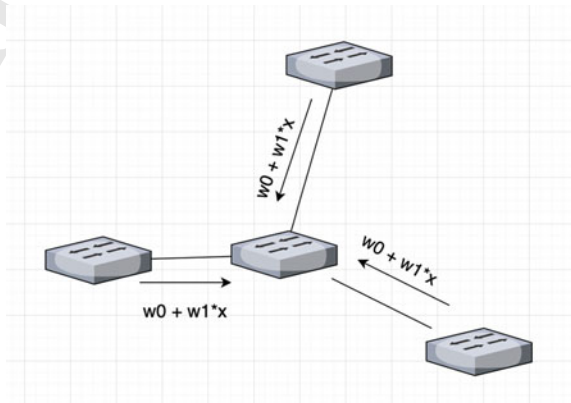
Regression Parameters Dissemination to Neighbours

Each sensing device sends information to the representative node in the neighbourhood. For example, if the device has learnt the linear model— $y = w_0 * x + w_1$, it will send the set of parameters— w_0, w_1 , the x_{min}, x_{max} ranges for the input data x and the number of rows of data— N it contains to the representative node (Fig. 11).

Generate Sampling Datasets

After the representative receives the required information from all neighbours n_i , the next step is to generate ‘sampling’ datasets— d_i for each set of local ML models received. The goal of this process is to generate input and output data pairs that can

Fig. 11 Neighbours sending their regression models



reproduce the datasets within the neighbouring nodes. Therefore avoiding transmission of the full datasets within the neighbouring nodes, which would be highly energy consuming. For each neighbouring sensing device n_i the process is the following:

- Take a random value x from the interval x_{min}, x_{max} . To do that, we need a uniform generator which gives us a uniformly distributed value p in the range $[0, 1]$. Then the random input x within the interval is:

$$x_{rand} = x_{min} + p * (x_{max} - x_{min}) \quad (12)$$

This is a uniformly generated input x from this interval.

- Before we calculate the output y we need to add Gaussian noise, in order to reduce overfitting and improve model generalisation. Gaussian noise is a value with zero mean and specific variance. We assume the variance, is relatively small, e.g., 0.1. Hence, we generate a Gaussian value q with $\mu = 0$ and $\sigma = 0.1$. Therefore, the final output looks like this: $y = w_0 * x + w_1 + q$.
- Calculate the output y .
- Generate N of these x, y pairs, run the PSO algorithm again and we will get the same model we received from node n_i . This proves that we have successfully, recreated the dataset in n_i , without needing to send its entire dataset.

The Fused PSO Model

In every node n_i , we combine the local data d_i with the aggregated generated 'sampling' datasets. The result is the fused dataset D which represents all the data of devices in the neighbourhood. We run the PSO algorithm again, this time on the fused dataset D . The final result is the fused regression model of that neighbourhood, which represents all the devices in that group $PSO D = f_i$.

After the generation of each fused model, they all need to be sent to the data center, which in turn has to pick the top- k performing models (in terms of energy consumption and prediction accuracy), for Predictive Modelling purposes. The end goal is to have at least k fused models outperforming the average and global models from the baseline solution so that they will be selected by the sink node.

In this chapter, we will assume that at least 70% of the nodes need to be represented by the top- k fused models so that they can represent the entire network. Therefore, a sensible value needs to be assigned to k , depending on the network type and the neighbourhood sizes. For a dense network $k = 1$ might be enough, however for a sparser network $k > 1$. The k values per network model will be set in the Performance and Comparative Assessment.

3.5 Network Model Impact

In the **Random Network** model, increasing the connectivity probability p increases the sizes of the neighbourhoods. Therefore, we expect larger neighbourhoods to have more accurate fused models, but have increased communication costs.

The **Small World** network is dependant on the order of nodes in the ring, the mean number of neighbours for each node and the rewiring probability β . Increasing the number of neighbours per node should guarantee more accuracy but at the cost of increased energy consumption. The rewiring probability and the order of the nodes, might add some unpredictability to this network model.

4 Implementation

This section discusses the software development aspects of this chapter in detail—the used datasets, the implementations of the Particle Swarm Optimisation algorithm and each of the network models.

4.1 Programming Language

The first task of the chapter was to choose a suitable programming language for the implementation. After some thought the circle was narrowed down to two programming languages—Java and Python. Some of the reasons being their ease of use, a large number of libraries available, object-oriented nature and the familiarisation at the time with those languages. Learning a new programming language would have introduced a lot of complexity and would have been time-consuming, therefore endangering the successful completion of the chapter. In the end, the chosen language was Python because of its suitability for quick prototyping, its versatility and convenience for plotting data, which is really important for analysing and evaluating results in this chapter.

Because of its versatility Python is dubbed as the Swiss Army Knife in the data science community [6]. However, the Standard Python Library alone was not sufficient for the implementation of this chapter. Additional third-party libraries were used for that additional functionality, that was missing in the standard module. SciPy and NumPy are one of the most used libraries in the Python data science community. They were useful, because of their large collection of high-level mathematical functions, ability to manipulate data and efficient operations on arrays [22]. The Matplotlib plotting library has played an important part in the chapter's performance assessment phase. This library made it straightforward to create quality graphics, which helped properly assess the performance of the machine learning models [25].



4.2 Data

Datasets Used

This chapter has used two datasets to distribute across the simulated sensing nodes

- Appliances Energy Prediction Dataset—this dataset was originally used to present and discuss data-driven predictive models for the energy use of appliances [2]. It includes measurements of temperature and humidity sensors from a wireless network. The readings are from 9 rooms in a house and the data has been collected at every 10 min for about 4.5 months. The house temperature and humidity conditions were monitored with a ZigBee wireless sensor network. Each wireless node transmitted the temperature and humidity conditions at every 3.3 min. Then, the wireless data was averaged for 10 min periods. Two random variables have been included in the dataset for testing the regression models and to filter out non-predictive attributes (parameters) [7].

For the purposes of the experiments in this chapter, each of the 9 rooms from the dataset is considered as a separate node with different temperature and humidity readings.

- GNFUV Unmanned Surface Vehicles Sensor Dataset - this dataset comprises 4 sets of mobile sensor readings data (humidity, temperature) corresponding to a swarm of four Unmanned Surface Vehicles (USVs) floating over the sea surface in a coastal area of Athens, Greece [10]. It contains readings about the humidity and temperature of a sea surface during two months—March and July. Each USV set contains records in the following format: {'USV-ID'; 'humidity-value'; 'temperature-value'; 'experiment-id'; 'sensing-time'}. The swarm of the USVs is moving according to a GPS predefined trajectory.

This dataset has been collected by 4 sensing devices for 2 different months. Therefore, using this dataset we can simulate $4 * 2 = 8$ nodes (Fig. 12).



Fig. 12 A picture of the USVs used in the experiments [10]

Data Diversity

After the datasets were downloaded, the next step was to analyse them and see if they are diverse enough for the purposes of the chapter. After all, the data had to be diverse, so that each node computed a unique set of regression coefficients, which is expected in a real wireless sensor network. The regression lines of the relationships between temperature and humidity (where the temperature is the independent variable and humidity the dependent one) in both datasets are seen in Fig. 13. From that, we can conclude that both datasets contain diverse linear regression models and each node would compute unique regression coefficients. As we can see from the above plots, in the Appliances dataset almost all sensing nodes have approximately the same median values of their temperature and humidity readings. This is not the case with the USV dataset, where there are more distinguishable differences between the medians. Overall, both datasets will be useful in the experimental phase of the chapter (Figs. 14, 15).

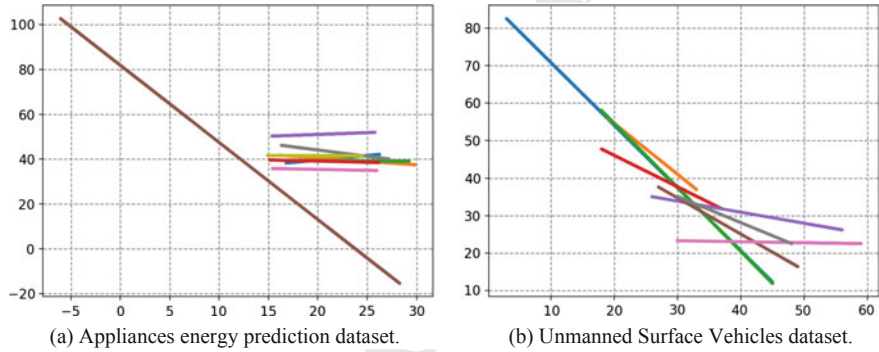


Fig. 13 Regression lines of the data within each sensing device

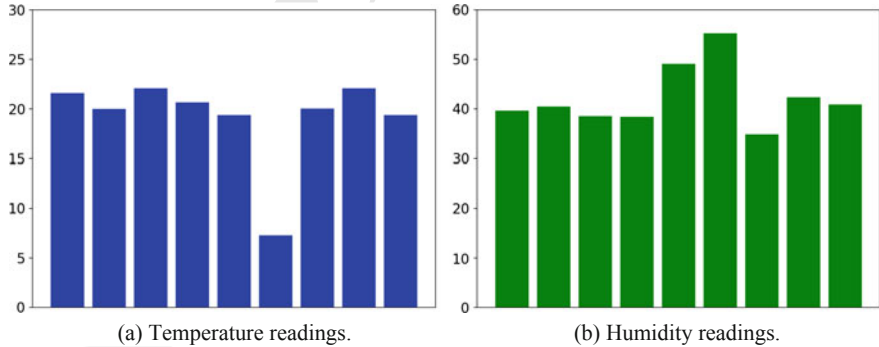


Fig. 14 Median values per node in the Appliances dataset

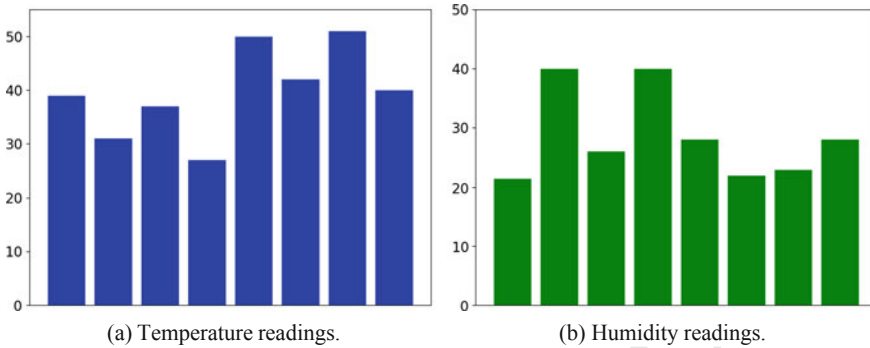


Fig. 15 Median values per node in the USV dataset

4.3 Particle Swarm Optimisation

In order to generate the local ML models, this chapter uses the Particle Swarm Optimisation (Swarm Intelligence) algorithm to calculate the local regression coefficients that best fit the data in each node.

4.3.1 Implementation

One of the numerous advantages of the algorithm is its ease of implementation. Therefore, implementing PSO was straightforward (see Algorithm 1). The algorithm maintains a swarm of particles, where each particle represents a potential solution. I took advantage of the object-oriented nature of Python to create a Swarm and Particle classes. The swarm class contains information that concerns all particles, such as global best fitness value and positions, the number of maximum iterations and particles (Fig. 16). It is responsible for updating the global best value and stopping the algorithm when it has converged. The particle class, as the name suggests, represents a single particle (see Fig. 17). It contains the boundaries of the search space, the position, current velocity, maximum velocity and the personal best position as fields.

```
class Swarm:
    def __init__(self, iters, stopping_condition, number_of_parameters):
        self.particles = []
        self.gbest = sys.maxsize
        self.gbest_pos = [0.0] * number_of_parameters
        self.iters = iters
        self.stopping_condition = stopping_condition
```

Fig. 16 Swarm class implementation


```
class Particle:
    def __init__(self, lower_boundary, upper_boundary, number_of_parameters):
        self.number_of_parameters = number_of_parameters
        self.pos = [0.0] * number_of_parameters
        self.lower_boundary = -60.0
        self.upper_boundary = 60.0
        self.pbest_pos = [0.0] * number_of_parameters
        self.pbest = -1
        self.v = [0.0] * number_of_parameters
        self.v_max = (self.upper_boundary - self.lower_boundary) * 0.05
        self.initialize_position()
```

Fig. 17 Particle class implementation

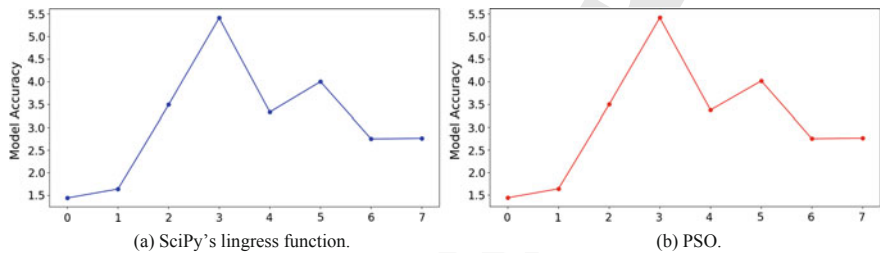


Fig. 18 Accuracy of generated regression coefficients for each sensing device's data

Its position is initialised as a uniform random value in the search space interval. The particle class looks like the following:

The objective function used in this chapter is the **Root Mean Square Error** (see 2.6) of the current position of the particle in the n-dimensional space (the parameters this particle represents) and the node's local dataset. It is evaluated for every particle at every iteration. All particles are aiming for the lowest value of the objective function (lower error means higher accuracy of the regression coefficients). Therefore the lowest RMSE value wins and the particle's parameters become the global best.

This implementation of the base PSO algorithm (without any optimisations or parameter tweaking) was tested on generating linear regression coefficients. It was tested against SciPy's linalg function, which also computes linear regression parameters. Figure 18 shows that their performances are exactly identical. This proves that while the algorithm may be very simple to implement, it produces fantastic results.

4.3.2 Algorithm Optimisations

Inertia Weight

Inertia weight adds a fraction of the last iteration's particle velocity to the current one. It was introduced by Shi and Eberhart as a mechanism to control the exploration

and exploitation abilities of the swarm, and as a mechanism to limit the maximum velocity a particle can reach and prevent all particles from leaving the search space [9]. Eberhart and Shi indicated that the use of the inertia weight w , which decreases linearly from about 0.9 to 0.4 during a run, provides improved performance in a number of applications [15]. This technique is used in this chapter and it enables the particles to explore a larger area at the start and as the algorithm progresses, slow down and converge to an optimum.

Velocity Clamping

Sometimes particles may reach high velocity and risk skipping appropriate search areas. They may jump over good solutions, and continue to search in fruitless regions of the search space. That is why particle velocity needs to be limited in order to find the balance between global exploration (explore different regions) and local exploitation (concentrate the search). If a particle's velocity exceeds a specified maximum velocity, the particle's velocity is set to that maximum velocity. Let $V_{max,j}$ denote the maximum allowed velocity in dimension j . If $V_{max,j}$ is too small, the swarm may not explore sufficiently beyond locally good regions [9]. On the other hand, too large values of $V_{max,j}$ risk the possibility of missing a good region. Particle velocity is adjusted before the position update using the following equation:

$$v_{i,j}(t+1) = \begin{cases} v_{i,j}(t+1) & \text{if } v_{i,j}(t+1) < V_{max,j} \\ V_{max,j} & \text{if } v_{i,j}(t+1) \geq V_{max,j} \end{cases} \quad (13)$$

4.3.3 Parameter Tuning

Swarm Size

The more particles there are in the swarm, the larger the search area that can be covered per iteration. This means that the search for optimal parameters may take fewer iterations. On the other hand, increasing particles increases greatly the computational complexity. The suitable swarm size mostly depends on the type of problem being solved. At the start of the chapter, a different number of particles has been experimented with, as can be seen from Fig. 19. Depending on the complexity of the regression model the number of particles needed to provide optimal accuracy may vary. The PSO implementation in this chapter uses between 50 and 100 particles depending on the regression models tested with (some models require particles to cover more searching space in order to find the optimal coefficients).

Number of Iterations

The number of iterations to reach a good solution is also problem-dependent. Too few iterations may terminate the search prematurely. A too large number of iterations has the consequence of unnecessary added computational complexity [9]. The maximum number of iterations needs to be set appropriately, in case another stopping condition

Fig. 19 PSO run with 30 iterations and different swarm sizes plotted against the resulting accuracy of the found parameters (linear regression)

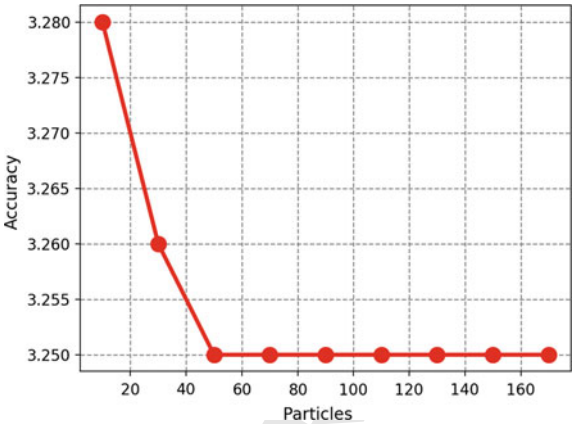
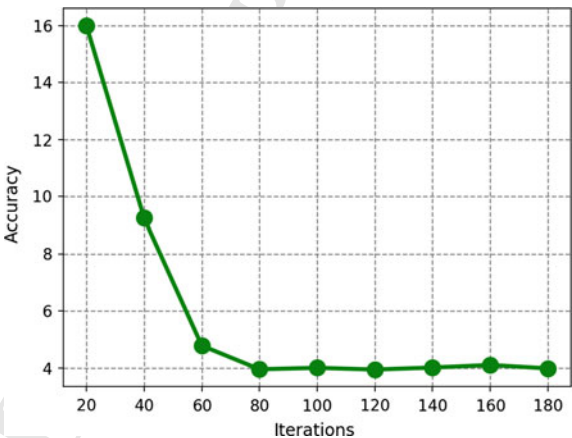


Fig. 20 PSO run with 50 particles and different maximum iterations (linear regression)



is not met. The maximum number of iterations experimented with varies from 30 to 70. As we can see from Fig. 20 the increasing number of iterations does not provide an increase in accuracy. Therefore, to avoid the additional computational overhead, for the task of finding the optimal regression coefficients, we just need to find the minimum number of iterations that provide the best accuracy.

4.4 Convergence

At the start, the PSO implementation had a fixed maximum number of iterations as the only terminating condition. The issue with that was after the particles had converged to a global minimum, the algorithm was still running and wasting iterations. An additional stopping condition was introduced to remove unnecessary iterations.

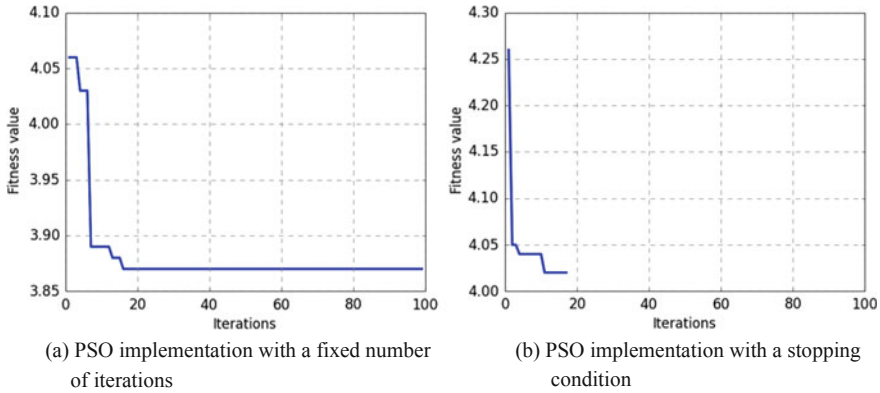


Fig. 21 **a** Shows that the algorithm has converged but is still iterating, **b** shows a smarter implementation with a stopping condition

The stopping condition used in this implementation is to terminate the algorithm when little improvement is observed in the particles' positions over a number of iterations. If the change in some particle positions is relatively small (less than a predefined s), this would mean that the swarm has converged to a solution. This solution introduces some complexity because sensible values need to be found for those two parameters: the window of iterations w and the difference in particle positions s . Therefore, if for every dimension in the n -dimension space the following statement is true for a particle p :

$$\|p.oldpos - p.newpos\| < \varepsilon \quad (14)$$

in consecutive w iterations, then we know the algorithm has converged (Fig. 21).

4.5 Network Models

Random Network

In the random network, every node has a probability p to be connected to every other node. The implementation of the random network requires an adjacency matrix, which generates the node neighbourhoods.

Algorithm 2: Pseudo code to generate a random network

Data: N an array of nodes p connectivity probability A adjacency matrix
begin
 for $i < size(N)$ **do**

```
654      for  $j < \text{size}(N)$  do
655          if  $i == j$  then
656              continue
657          end
658          if  $r \text{ and}() < p$  then
659               $A[i][j] = 1$ 
660          Else
661               $A[i][j] = 0$ 
662          end
663      end
664  end
665  end
```

Figure 22 shows the implemented adjacency matrix. The random network takes in as parameters the dataset and connectivity probability. After that, it initialises the adjacency matrix and looks it up for every node to assign its neighbours.

Small World Network

The NetworkX python library has been used to obtain an implementation of the Watts-Strogatz network. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks [3]. The parameters that the watts_strogatz_graph() method takes in are listed in Table 2.

Fig. 22 Adjacency matrices for random networks with different connectivity probabilities

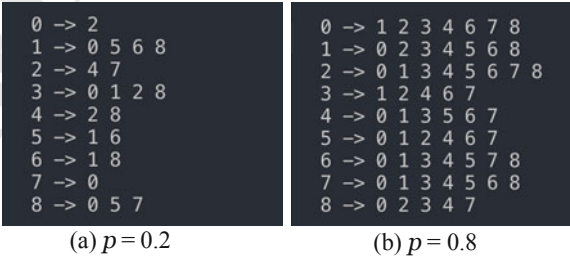


Table 2 Parameters needed to obtain a small world network from the NetworkX python library	
Parameter	Description
n	Number of nodes
k	Each node is joined with k nearest neighbours in the ring topology network
p	Probability of rewiring each edge

The method first creates a ring over n nodes. Then each node in the ring is joined with its k nearest neighbours (or $k - 1$ neighbours if k is odd). Then new edges are created by replacing some already existing edges as follows: for each edge (u, v) in the underlying “ n ring with k nearest neighbours” with probability p replace it with a new edge (u, v) with uniformly random choice of existing node w [3].

5 Performance and Comparative Assessment

This section goes into detail how well the proposed methodology performs against the baseline solution and discusses the pros and cons of each network model. Each network type is evaluated in two cases: when nodes generate linear and nonlinear regression coefficients.

5.1 Prerequisites

Types of Models

This section will compare the performance of the proposed fused model (discussed in detail in Sect. 3.4) against 4 other types of models:

- **Global Model**—all sensing devices send their entire datasets to the remote sink, which in turn generates a regression model based on all the data.
- **Average Model**—all sensing devices run the PSO algorithm to generate their local regression models. After that, every device sends their model to the sink node, which takes the average of them.
- **Local-Global Model**—similar to the Global model, but just for a specific neighbourhood—all the nodes from the neighbourhood send their entire datasets to the representative node of the neighbourhood, which generates a regression model based on the received data plus its local data.
- **Local-Average Model**—similar to the Average model, but just for a specific neighbourhood—all nodes from the neighbourhood generate their local regression models. Each device sends its local model to the representative of the neighbourhood, which takes the average of all the received models plus its local model.

Linear and Non-linear Regression Models

The proposed methodology in this chapter has experimented with linear and nonlinear regression models. In the linear case, both networks were tested with simple linear regression models of the form: $y = w_0 x + w_1$, where x is the temperature reading and y is the humidity. In the other case, several nonlinear regression models have been experimented with but for the purposes of the performance assessment, we will



use the one that was most accurate for the two datasets used in this chapter—the **Misrald** model. It takes the following form:

$$y = \frac{\beta_1 \beta_2 x}{1 + \beta_2 x}, \quad (15)$$

where again x is the temperature and y is the humidity data.

Evaluation

In order to evaluate the proposed fused models, we need to calculate their accuracy and energy cost and compare their performance against the other model types. The energy consumption is calculated using the energy cost of the Mica2 instructions mentioned in Table 1. Moreover, the energy consumption of the global and average models is calculated with respect to the average diameter of the network (the number of hops the packets need to go through to reach the sink node). Moreover, we assume that the communication between nodes of the same neighbourhood is direct (0 hops). Meanwhile, the accuracy of each model is evaluated using a test dataset that consists of a 5% random sample of the full dataset (4.2). Depending on the network type, the experiments will show if the top- k performing models are the fused models (k depends on the connectivity of the network and the network model itself).

5.2 Random Network Assessment

This section will discuss the results of the proposed methodology when run in a random network. The connectivity probabilities experimented with range from 0.1 to 0.9. The diameter of the random network is $d = \frac{n * p}{\log n}$, where n is the number of nodes and p is the connectivity probability [26]. Therefore, when a device wants to send packets to the sink node, they would have to go through d other nodes, to reach their destination. This means that these d nodes, as well as the source node will pay those communication costs. When the connectivity probability increases, so do the neighbourhoods and sensing devices need to send their regression coefficients to more neighbours. This is illustrated in Fig. 23. The energy cost of the fused models is lowest at $p = 0.1$ and highest at the opposite end— $p = 0.9$.

Depending on the connectivity of the network, the data center needs to pick a different number of fused models k . The increase in p reduces the top- k models that need to be selected, in order to represent at least 70% of the devices in the network. Figure 24 shows that increasing the density of the network, decreases the top- k models, the sink node needs.

Linear Regression Models

In this case, the PSO algorithm will be used to generate linear regression coefficients. As we can see from Fig. 25 as the connectivity in the random network increases, the fused linear models become more accurate, which is to be expected, since adding



Fig. 23 Connectivity probability and energy cost of the fused models

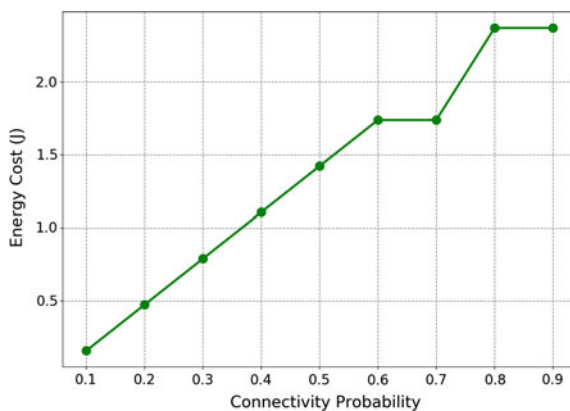


Fig. 24 Number of top-k models per connectivity probability

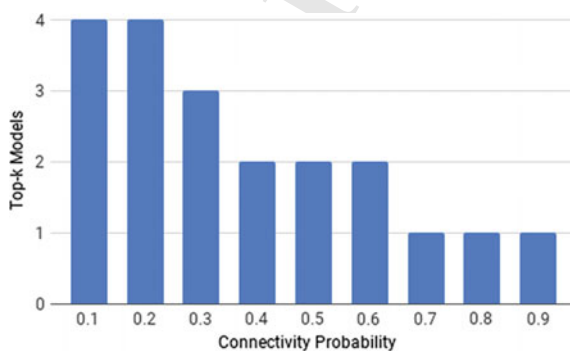
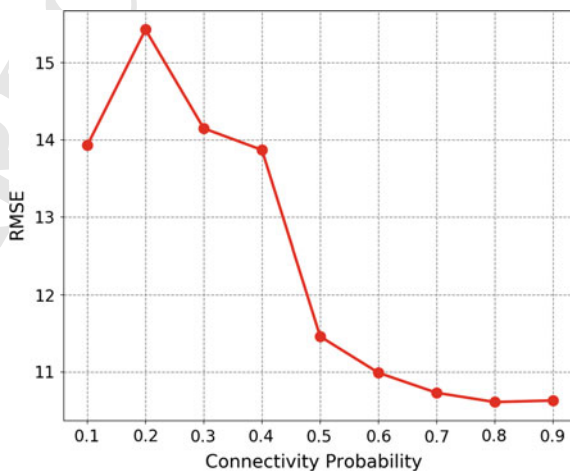


Fig. 25 Connectivity probability and median accuracy of the linear fused models



more nodes to the neighbourhoods means adding more local data to train the models. After $p = 0.7$, there is no significant improvement in the prediction accuracy, therefore we can assume that that is our optimal p value. Therefore, from Figs. 23 and 25 we can conclude that we can sacrifice that little extra energy consumption to achieve relatively greater accuracy. If we assume that $p = 0.7$ is the optimal value the following plot shows the result of the experiment for each model type.

Firstly, we can see that each model type is clustered into a group and there are almost no outliers. The Global model is the most energy consuming and the local average models are least accurate. In terms of prediction accuracy, the proposed fused models are on par with the local-global and Global models, which are our point of reference for accuracy, since they are trained on the entire dataset in the neighbourhood/network. At the same time, the fused models are two orders of magnitude more energy efficient. Some local-average models consume even less energy but have, on average, 24.4% higher error estimate. From Fig. 24 it follows that for $p = 0.7$ the sink node would select only the best performing model. Therefore, if the sink is looking for the best accuracy-energy cost tradeoff, it would pick one of the fused models (Fig. 26).

Nonlinear Regression Models

This time, each device in the Random network will compute its local nonlinear regression model. As we can see from the plot above, the results are much different from the ones in the linear case. We cannot make out any trends in the accuracy of the nonlinear fused models and there is no evident relationship between connectivity probability and prediction accuracy in this case. Therefore, we can use the p value where the fused accuracy is best -0.6 and use that to test how the proposed methodology performs in this case (Fig. 27).

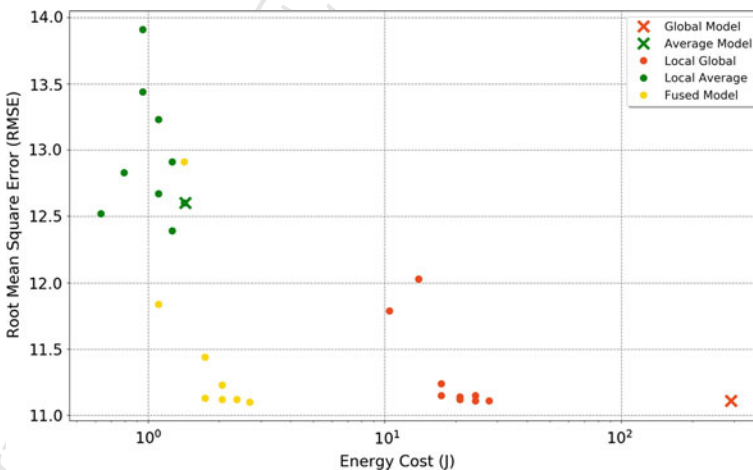
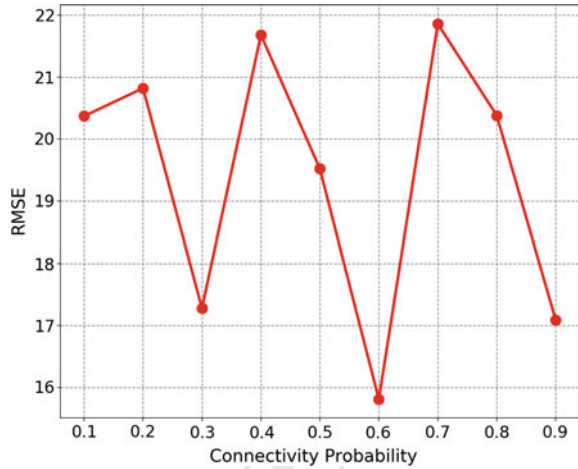


Fig. 26 Accuracy and energy consumption of each linear model in a Random network with connectivity probability = 0.7

Fig. 27 Connectivity probability and median accuracy of the nonlinear fused models



From Fig. 28 it follows that when each node is calculating nonlinear regression coefficients, there are more outliers. Half of the fused models have the same prediction accuracy as the ‘naive’ average models. On the contrary, the other half have on average 30% higher prediction accuracy. In that regard, they are on par with the global models, but at the same time, they consume between 10 and 100 times less energy. Some local-average models consume a little less energy but are much more inaccurate. In this case, the connectivity probability is equal to 0.6. Consequently, the sink node has to select *top k* models where $k = 2$. Overall, there are more than 2

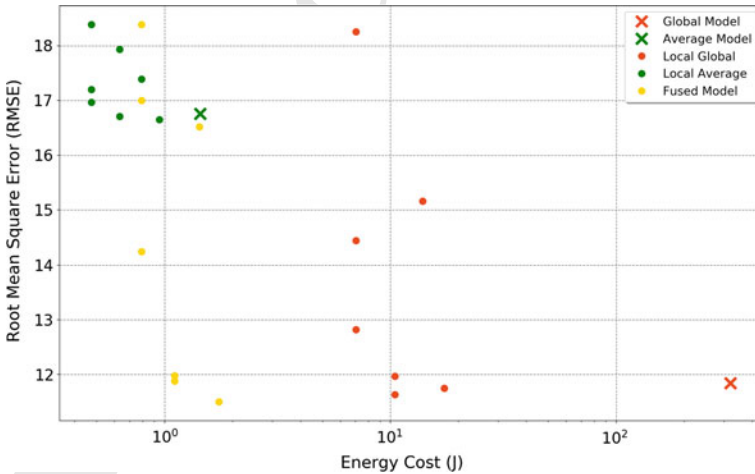


Fig. 28 Accuracy and energy consumption of each nonlinear model in a Random network with connectivity probability = 0.6

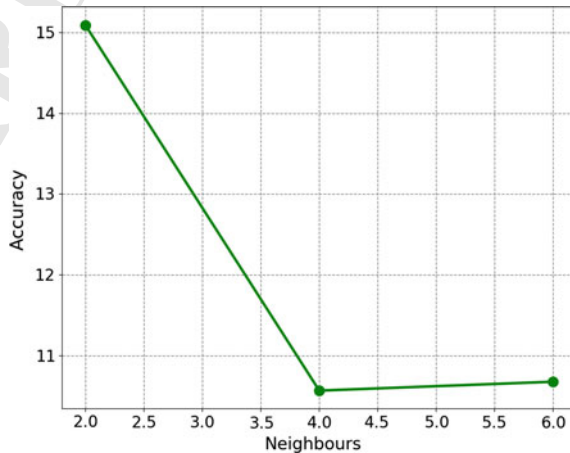
fused models that by far outperform the baseline solution. Therefore, this experiment is successful.

5.3 Small World Network Assessment

The datasets used in this chapter contain data for 8 (USV dataset) and 9 sensing devices (Appliances dataset). A suitable number of neighbours had to be chosen in order to appropriately assess the small world network. At the start, every device is connected to its k closest neighbours ($k - 1$ if k is odd). Therefore, every node needs to have an even number of neighbours and for the given datasets, the options are 2, 4 and 6. Increasing the neighbours per node from 2 to 4 lowers the fused model error value by 30%. Having 6 neighbours does not provide any increase in the fused model accuracy and it further increases the communication costs. Therefore, 4 neighbours will lead to a great increase in the models' accuracy, without being energy expensive. Consequently, for the purposes of this assessment, we will choose the neighbours per node to be 4 (Fig. 29).

The number of top performing models— k needs to be established for this network model. The number of neighbours— N , is the parameter that controls the size of the neighbourhoods. If we set $N = 4$, for the purposes of the experiments, it follows that the mean size of the neighbourhoods will be 5. If the datasets that are used in this chapter have been collected from 8 and 9 sensing devices, then one neighbourhood will consist of 62.5 and 55.6% of all devices respectively. Therefore, the data center will need $k = 2$ models to involve at least 70% of all the nodes' data in the generation of the models. Consequently, for the proposed methodology to be successful in this network model, the top 2 best performing regression models have to be fused models.

Fig. 29 The fused model accuracy for a specific number of neighbours



The small world network offers “six degrees of separation”, that means the maximum distance between any 2 nodes is 6 hops. This is true for social media as well—everyone in the world is connected to everyone else by no more than six degrees of separation [16]. This property will be useful when calculating the energy consumption of the Global and Average models because the packets from the source devices would have to go through 6 other devices (in the worst case) before reaching their destination - the sink node.

Linear Regression Models

In this case, the sensing and computing devices in the small world network will have to compute their local linear regression models. As we can see from the above plot we cannot make a conclusion about the relationship between the fused model prediction accuracy and the rewiring probability of the network. For that reason, for the purposes of the experiment, we will use the rewiring probability for which the fused error is minimum—0.6. The models are mostly grouped together in clusters (Figs. 30 and 31).

In terms of accuracy, the fused models perform as good as the global models and some even better. Moreover, in comparison to the average models, most fused models are on average 30% more accurate. The energy cost of the fused models is decent, most of them are on par with the Average model or better and they use 2 orders of magnitude less energy than the Global model. In this case, the sink node will have to pick the 2 best models and they would be the proposed fused models. As a result, the proposed methodology in this experiment is successful.

Nonlinear Regression Models

This time, the proposed methodology will generate nonlinear local regression coefficients. As in the linear case, we cannot infer any relationship between the rewiring probability value and the fused model accuracy. The rewiring value for which the

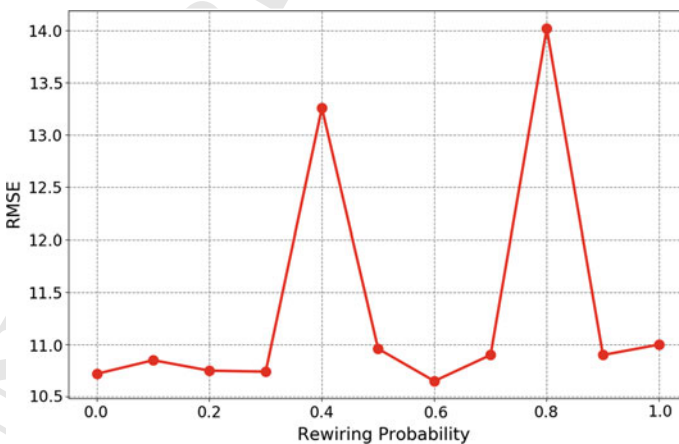


Fig. 30 Rewiring probability and median accuracy of the linear fused models

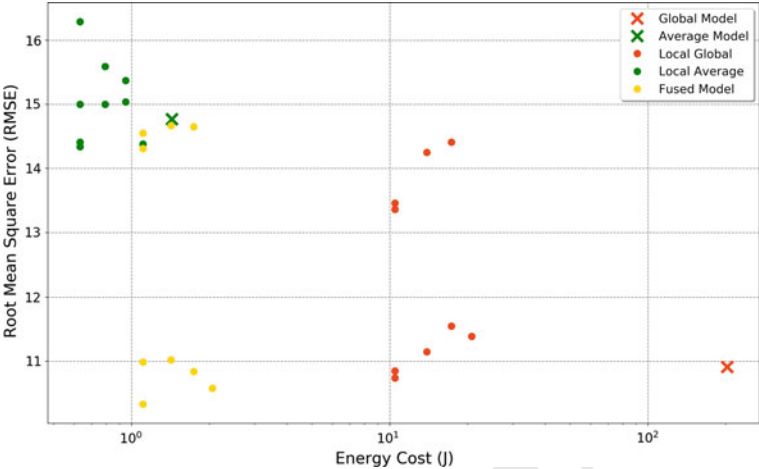


Fig. 31 Accuracy and energy consumption of each linear model in a Small World network with rewiring probability = 0.6

fused RMSE is minimum is 0.8 and this will be used in the experiment. As we can see the proposed methodology for the small world networks is not as effective with nonlinear regression models. This time, the Average model is more accurate than the Global model and much more energy efficient, therefore it is the point of reference for evaluating the proposed methodology in this case. The fused models' energy efficiency is on par with the Average model, but only one of them performs as good, in terms of prediction accuracy. When the sink node evaluates the top 2 performing models only one of them will be a fused model. Consequently, the experiment is unsuccessful, as the fused models do not address the limitations of the baseline solution. Therefore, when the small world network is run with nonlinear regression models, the proposed methodology is not effective (Figs. 32 and 33).

6 Conclusions and Future Work

This section summarises the chapter, the results of the performance assessment and discusses possible future improvements.

6.1 Assessment Results

Random Network Model

This network model has only one parameter - connectivity probability. Increasing the connectivity, resulted in increased fused model accuracy and energy consumption.



Author Proof



Author Proof

Author Proof

Author Proof

In the nonlinear case, there was no relationship between the connectivity probability and the prediction accuracy of the fused models, but their accuracy was best at $p = 0.6$ and that was the value used for the experiment. This time, there were some outliers, which achieved the same prediction error as the ‘naive’ Average model. On the other hand, some of the fused models had the same accuracy as the Global model and used 2 orders of magnitude less energy. Overall, the experiment was successful. Overall, both with linear and nonlinear regression models, the fused models were outperforming the baseline solution. Therefore, we can conclude that the Random network was successful in adopting the proposed methodology of this chapter.

Small World Network Model

The parameters of this network model are the number of neighbours and the rewiring probability. Increasing the neighbours, resulted in increased fused model accuracy and energy cost, as expected. For the purposes of the experiments, the number of neighbours was set to 4 (offered best energy consumption-prediction accuracy value). When run with linear regression models, there was no evident relationship between the rewiring probability and the energy consumption or accuracy of the models. Therefore, the value, for which the fused model error was lowest, was selected for the experiment— $p = 0.6$. Again, there were some outliers, but roughly 56% of the fused models were outperforming the baseline solution, combining Global model accuracy and Average model energy efficiency.

In the other case, once more, there was no connection between the rewiring probability of the network and the accuracy of the fused models. The accuracy was best at $p = 0.8$ and this value was used for the experiment. The proposed methodology was not as effective in this case. This time, the Average model had a lower prediction error than the Global model and because of its energy efficiency, it was the point of reference for evaluating the fused models. The fused models’ energy efficiency was on par with the Average model, but only one of the proposed models had higher prediction accuracy. Therefore, in this case, the proposed methodology was not effective and did not outperform the baseline solution.

Overall, the Small World network proved more unpredictable, compared to the Random network model. It was dependent on the rewiring probability (was largely responsible for the random behaviour of the network) and the positions of the nodes in the ring structure at the initialisation phase. This is why the position of the nodes in the initial ring structure was randomised at each run, to reduce bias as much as possible. The proposed methodology was efficient when the devices were generating linear regression models. It was evident that most of the fused models outperformed the baseline solution. This was not the case when run with nonlinear coefficients. At most only one fused model was performing on par with the baseline solution, all others had worse prediction accuracy. Therefore, we can conclude that the Small World network was successful in adopting the proposed methodology but only when generating linear regression coefficients.

6.2 Future Work

Should this chapter be extended in the future, this section proposes some suggestions for future work.

Larger Datasets

The chapter has experimented with datasets that have been generated from 8 and 9 sensing nodes. On the other hand, a real-life wireless sensor network can have from a few to several hundred or even thousands of sensing nodes. Therefore, it might be appropriate in the future for the experiments to use datasets that have been generated from a greater number of nodes. That will result in more realistic experiments and will test the scalability of the proposed methodology.

Network Models

The current approach experiments with only 2 network models—Random network and Small World network. Future extensions of this chapter may experiment with a larger set of network models.

Parameter Estimation Algorithms

The Particle Swarm Optimisation algorithm is used to generate the optimal regression coefficients per device. Experimenting with other regression estimation methods might be useful such as Least Squares, Gradient Descent and Regularisation. This would allow to compare the performance of the different algorithms and choose the most appropriate for the purpose of this chapter.

Real Sensing Devices

This chapter generated software that only simulates sensing devices and the communication between them. In the future, one might experiment with real IoT devices such as Raspberry Pi and see if it produces the same results.

More Nonlinear Models

Six nonlinear regression models have been experimented with, to see which one is the most appropriate for the used datasets. In the future, one might experiment with a wider range of nonlinear regression models, to produce more accurate local regression models.

6.3 Summary

This chapter has presented a lot of challenges and required a significant amount of research, especially at first. The following was achieved during the course of the chapter:

- Implemented and optimised the Particle Swarm Optimisation algorithm.
- Proved that PSO is very effective at calculating optimal regression parameters.
- Implemented two network models, in order to exploit node networking neighbourhoods.
- Proved that the proposed methodology of this chapter is effective at addressing the limitations of the baseline solution.

There are several conclusions we can draw from this chapter. Firstly, the PSO algorithm is a reliable parameter estimation approach. It exhibits a rapid convergence tendency and converged after at most 30–35 iterations, depending on the complexity of the regression model. When calculating the linear regression coefficients the results from the algorithm were identical to SciPy's `linregress` function [14]. Özsoy and Örkücü [23] prove in their paper that PSO is successful in estimating nonlinear regression parameters, as well.

Exploiting neighbourhoods significantly reduces the number of nodes communicating with the Data Center. It also solves the scalability issues of the baseline solution. This is because only one representative per neighbourhood sends the fused model to the sink, instead of every single node sending their local model or data. Depending on the connectivity of the network, the data center will pick the top- k fused models to use for predictive modelling. When the network is dense, the sink node will need less k models.

Overall, the proposed methodology is an efficient mechanism to tradeoff communication overhead and accuracy. It tackles the limitations of the baseline solution the expensive communication costs (Global model) and the decreased prediction accuracy, because of the naive averaging of regression models (Average model).

Acknowledgements This research is funded by the EU-H2020 GNFUV Project (#Grant 645220) and the EU-H2020 MSCA INNOVATE Project (#Grant 745829).

References

1. Anagnostopoulos, C., Hadjiefthymiades, S.: Advanced principal component-based compression schemes for wireless sensor networks. *ACM Trans. Sen. Netw.* **11**(1), 7:1–7:34 (2014). ISSN 1550-4859
2. Candanedo, L.M., Feldheim, V., Deramaix, D.: Data driven prediction models of energy use of appliances in a low-energy house. *Energy Build.* **140**, 81–97 (2017)
3. Aric Hagberg, P.S., Dan Schult. *Networkx*. <https://networkx.github.io/> (2014). Cited 29 Oct 2018
4. Barabási, A.-L.: *Network science* acknowledgements random networks. Creative Commons (2014)
5. Beers, B.: *What regression measures*. Investopedia (2019). Cited 2 Feb 2019
6. Bhatia, R.: *Why do data scientists prefer python over java?* Analytics India Magazine (2018). Cited 27 Feb 2019
7. Candanedo, L.M., Feldheim, V., Deramaix, D.: Data driven prediction models of energy use of appliances in a low-energy house. *Energy Build.* **140**, 81–97 (2017). ISSN 0378-7788. doi: <https://doi.org/10.1016/j.enbuild.2017.01.083>

8. Newman, M.E.J., Watts, D., Strogatz, S.H.: Random graph models of social networks. *Proc. Natl. Acad. Sci. USA* **99**(1), 2566–2572 (2002). <https://doi.org/10.1073/pnas.012582999>
9. Engelbrecht, A.P.: *Particle Swarm Optimization*. Wiley (2007)
10. Harth, N., Anagnostopoulos, C.: Edge-centric efficient regression analytics. <http://eprints.gla.ac.uk/160937/>. April 2018
11. M. Incorporated.: Predictive modeling: The only guide you need. <https://www.microstrategy.com/us/resources/introductory-guides/predictive-modeling-the-only-guide-you-need> (2018). Cited 23 March 2019
12. Indu, S.D.: Wireless sensor networks: Issues and challenges. *Int. J. Comput. Sci. Mob. Comput.*, 681–685 (20140)
13. Jj and Jj. MAE and RMSE—which metric is better?—human in a machine world—medium. <https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d> (2016). Cited 1 Oct 2018
14. Jones, E., Oliphant, T., Peterson, P. et al.: *SciPy: Open source scientific tools for Python*. <http://www.scipy.org/> (2001)
15. A. Kaveh. *Particle Swarm Optimisation*, chapter 2. Springer International Publishing, 2014
16. Keith: The history of social media: Social networking evolution! <https://historycooperative.org/the-history-of-social-media/journal=HistoryCooperative> (2019). Cited 21 March 2019
17. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: *Proceedings of ICNN'95—International Conference on Neural Networks*, vol. 4, pp. 1942–1948 (1995). <https://doi.org/10.1109/icnn.1995.488968>
18. Lueth, K.L.: State of the IOT 2018: Number of IOT devices now at 7b—market accelerating. *IoT Analytics* (2018). Cited 28 Feb 2019
19. Lv, Y., Tian, Y.: Design and application of sink node for wireless sensor network. In: *2010 2nd International Conference on Industrial and Information Systems*, vol. 1, pp. 487–490 (2010)
20. Math.com: Graphing equations and inequalities—slope and y-intercept—in depth. <http://www.math.com/school/subject2/lessons/S2U4L2DP.html> (2000–2005). Cited 24 March 2019
21. Mehl, B.: 6 IOT communication protocols for web connected devices. Kisi. <https://www.getkisi.com/blog/internet-of-things-communication-protocols> (2018). Cited 1 March 2019
22. Oliphant, T.: *Numpy: A guide to numpy*. USA: Trelgol Publishing (2006). Cited 24 March 2019
23. Özsoy, V.S., Örkücü, H.: Estimating the parameters of nonlinear regression models through particle swarm optimization. *Gazi Univ. J. Sci.* **29**, 187–199 (2016)
24. Pascual: Understanding regression error metrics. <https://www.dataquest.io/blog/understanding-regression-error-metrics/> (2018). Cited 20 Feb 2019
25. Python, R.: Python plotting with matplotlib (guide)—real python. <https://realpython.com/python-matplotlib-guide/> (2018). Cited 11 Oct 2018
26. Riordan, O., Wormald, N.: The diameter of sparse random graphs. *Combinat. Prob. Comput.* **19**(5–6), 835–926 (2010)
27. Srivastava, T.: 7 important model evaluation error metrics everyone should know. <https://www.analyticsvidhya.com/blog/2016/02/7-important-model-evaluation-error-metrics/> (2017). Cited 24 Nov 2018
28. Swaminathan, S.: Linear regression—detailed view. <https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86> (2018). Cited 21 Oct 2018
29. V. S. Technology: What are mean squared error and root mean squared error? <https://www.vernier.com/til/1014/> (2001). Cited 1 Oct 2018
30. Montoya, S.: How to calculate the Root Mean Square Error (RMSE) of an interpolated pH raster? <https://www.hatarilabs.com/ih-en/how-to-calculate-the-root-mean-square-error-rmse-of-an-interpolated-ph-raster> (2018). Cited 20 March 2019
31. T. P. S. University: 15.5—nonlinear regression. <https://newonlinecourses.science.psu.edu/stat501/node/370/> (2018). Cited 13 Jan 2019
32. Wikipedia: Watts-strogatz model https://en.wikipedia.org/wiki/Watts-Strogatz_model (2018). Cited 3 Nov 2018



- 1022 33. Wikipedia: Particle swarm optimization [https://en.wikipedia.org/wiki/Particle_swarm_opti-](https://en.wikipedia.org/wiki/Particle_swarm_optimization)
1023 [mization](https://en.wikipedia.org/wiki/Particle_swarm_optimization) (2018). Cited 30 Oct 2018
1024 34. Wikipedia: Linear regression https://en.wikipedia.org/wiki/Linear_regression (2019). Cited 8
1025 Oct 2018
1026 35. Wikipedia: Random graph https://en.wikipedia.org/wiki/Random_graph (2019). Cited 19 Jan
1027 2019

Author Queries

Chapter 7

Query Refs.	Details Required	Author's response
AQ1	Please check and confirm if the inserted citation of Figs. 1–6, 8, 9, 11, 12, 14–16, 21, 26, 27, 29–33 are correct. If not, please suggest an alternate citation. Please note that figures should be cited sequentially in the text.	

MARKED PROOF

Please correct and return this set

Please use the proof correction marks shown below for all alterations and corrections. If you wish to return your proof by fax you should ensure that all amendments are written clearly in dark ink and are made well within the page margins.

<i>Instruction to printer</i>	<i>Textual mark</i>	<i>Marginal mark</i>
Leave unchanged	... under matter to remain	Ⓟ
Insert in text the matter indicated in the margin	⋏	New matter followed by ⋏ or ⋏ [Ⓢ]
Delete	/ through single character, rule or underline or ⌞ through all characters to be deleted	Ⓞ or Ⓞ [Ⓢ]
Substitute character or substitute part of one or more word(s)	/ through letter or ⌞ through characters	new character / or new characters /
Change to italics	— under matter to be changed	↵
Change to capitals	≡ under matter to be changed	≡
Change to small capitals	≡ under matter to be changed	≡
Change to bold type	~ under matter to be changed	~
Change to bold italic	≈ under matter to be changed	≈
Change to lower case	Encircle matter to be changed	≡
Change italic to upright type	(As above)	⋏
Change bold to non-bold type	(As above)	⋏
Insert 'superior' character	/ through character or ⋏ where required	Y or Y under character e.g. Y or Y
Insert 'inferior' character	(As above)	⋏ over character e.g. ⋏
Insert full stop	(As above)	⊙
Insert comma	(As above)	,
Insert single quotation marks	(As above)	Y or Y and/or Y or Y
Insert double quotation marks	(As above)	Y or Y and/or Y or Y
Insert hyphen	(As above)	⌞
Start new paragraph	⌞	⌞
No new paragraph	⌞	⌞
Transpose	⌞	⌞
Close up	linking ○ characters	○
Insert or substitute space between characters or words	/ through character or ⋏ where required	Y
Reduce space between characters or words		↑