Enhancing Prediction in Cyclone Separators through Computational Intelligence

Oluwaseyi Ogun
Process Systems Engineering Laboratory
Obafemi Awolowo University
Ile-Ife, Nigeria
ogun.o.a@keemail.me

Mbetobong Enoh
Electrical and Electronics Engineering
Akwa Ibom State University
Nigeria
mbetobongenoh@aksu.edu.ng

Georgina Cosma
Department of Computer Science
Loughborough University
Loughborough, United Kingdom
g.cosma@lboro.ac.uk

Aboozar Taherkhani
School of Science and Informatics
De Montfort University
Leicester, United Kingdom
aboozar.taherkhani@dmu.ac.uk

Vincenzo Madonna
PEMC Research Group
University of Nottingham
Nottingham, United Kingdom
Ezzvm2@nottingham.ac.uk

***Abstract*—Pressure drop prediction is critical to the design and performance of cyclone separators as industrial gas cleaning devices. The complex non-linear relationship between cyclone Pressure Drop Coefficient (PDC) and geometrical dimensions suffice the need for state-of-the-art predictive modelling methods. Existing solutions have applied theoretical/semi-empirical techniques which fail to generalise well, and the suitability of intelligent techniques has not been widely explored for the task of pressure drop prediction in cyclone separators. To this end, this paper firstly introduces a fuzzy modelling methodology, then presents an alternative version of the Extended Kalman Filter (EKF) to train a Multi-Layer Neural Network (MLNN). The Lagrange dual formulation of Support Vector Machine (SVM) regression model is also deployed for comparison purposes. For optimal design of these models, manual and grid search techniques are used in a cross-validation setting subsequent to training. Based on the prediction accuracy of PDC, results show that the Fuzzy System (FS) is highly performing with testing mean squared error (MSE) of 3.97e-04 and correlation coefficient (R) of 99.70%. Furthermore, a significant improvement of EKF-trained network (MSE = 1.62e-04, R = 99.82%) over the traditional Back-Propagation Neural Network (BPNN) (MSE = 4.87e-04, R = 99.53%) is observed. SVM gives better prediction with radial basis kernel (MSE = 2.22e-04, R = 99.75%) and provides comparable performance to universal approximators. Of the conventional models considered, the model of Shepherd and Lapple (MSE = 7.3e-03, R = 97.88%) gives the best result which is still inferior to the intelligent models.**

***Index Terms*—Cyclone, Pressure drop coefficient, Fuzzy system, Support vector machine, Extended Kalman filter, Multi-layer neural network, Cross-validation.**

I. Introduction

Environmental pollution is a critical concern in many nations of the world due to its adverse effect on nature, and ultimately on lives. Many industrial activities release large gas effluents with particulate contaminants which must be removed before discharge.

Cyclone separators are industrial air pollution control devices for separating entrained particles from carrier gas streams before they are discharged. Besides the design simplicity and cost effectiveness of these devices, the lack of moving parts makes maintenance very cheap, and hence very attractive to the industry [1]. Some industrial applications include the removal of coal dust in power plants, removal of saw dust in sawmills, as spray dryers, etc. However, cyclone separators, like any other technology, require improvements to enhance performance and thus promote a wider adoption and with the deleterious effect of particulate pollution, it becomes imperative to enhance the performance of pollution control systems [2]. Numerous designs of cyclones exist which are in use for different purposes such as the uniflow, straight-through and reverse flow cyclones. However, the reverse flow cyclone is more commonly used for industrial gas cleaning (Fig. 1). Cyclones exploit the centrifugal force generated by the circular spinning of the inlet gas stream at high speed to bring about separation. This then causes the particulate solids to be thrown to the cyclone walls, loose speed and subsequently fall to the bottom where they are discharged. Besides particle collection efficiency, pressure drop is another major performance metric for performance evaluation in cyclones, which relates to the energy of the inlet gas stream. To this end, accurate mathematical modelling of the complex relationship between the pressure drop and cyclone dimensions is critical. Several approaches exist in the literature to describe the effect of cyclone geometry on pressure drop. These include: mathematical models [3]; theoretical or semi-empirical models (e.g., [4][5]); statistical models (e.g., [6]); and Computational Fluid Dynamics (CFD) simulations, (e.g., [7]).

Theoretical or semi-empirical mathematical models based on physical descriptions and detailed understanding of gas flow patterns and energy loss mechanisms in cyclones have been developed by several researchers, such as the models of [4][8][9]. However, different simplifying assumptions inherent in these models, and different physical principles being exploited in the modelling can result in significant

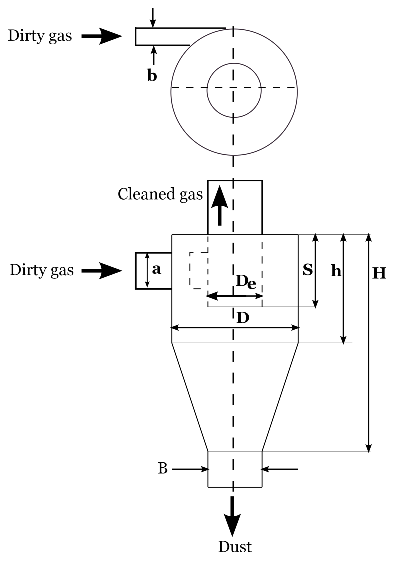

Fig. 1. Schematic of a reverse-flow cyclone separator

disparities between predictions and measurements. For example, [10] stated that none of these models predicts pressure drop accurately for a wide range of cyclone designs, and pressure drop predictions of some models are twice the measured values [11]. Computational Fluid Dynamics (CFD) simulations provide a reliable method to examine the effect of design changes on performance and provide an excellent approach to modelling cyclones as they are able to predict fluid flow patterns in great details [12]. However, CFD can be computationally challenging particularly in solving the complex mathematics of the famous Navier-Stokes equations that result. While data-driven intelligent methods such as the Artificial Neural Network (ANN) [13] and the Least Squares Support Vector Machine (LS-SVM) [3] are promising and have yielded superior modelling capabilities, there is still ample room for improvements, specifically with the choice of learning algorithm, network architecture and hyper-parameters selection/optimisation.

The current state-of-the-art development is hinged on Artificial Intelligence (AI), and has motivated the need for data-driven intelligent methods which can effectively extract useful information from domain data for decision making. Thus, towards an efficient design which minimises the pressure drop of inlet gas stream through the separator for effective gas cleaning, this study develops intelligent pressure drop prediction models. Due to the complex non-linearity between *PDC* and cyclone dimensions, existing approaches have not provided good generalisation of pressure drop predictions, thus motivating the need for improved neural and fuzzy *PDC* predictions, which are especially suited to model complexities and highly non-linear trends to arbitrary accuracy. Therefore, this paper introduces a fuzzy model and a more effective learning algorithm over the traditional Back-Propagation Neural Network (BPNN), for pressure drop prediction in cyclone separators.

The aim of this paper is two-fold: first, to use the dataset from the measurement of ninety-eight cyclone configurations [10] to develop prediction models, and then to draw conclusions from the obtained models for the more suitable approach (based on results) to pressure drop prediction in cyclone separators. The paper is structured as follows: Section II discusses the experimental methodology which include the process of selecting the variables for building the prediction models; Section III develops the mathematical models and algorithms for the prediction models; Section IV presents simulation results with discussion, and comparison of models and finally, conclusions are presented in Section V.

II. Experimental Methods

Selection of the right response and predictor variables for cyclone modelling is critical to a successful prediction, and the suitability of the consequent model for other purposes will greatly depend on the variables used. Thus, it has been shown that all dimensions of a cyclone separator in Fig. 1 affect its pressure drop to different extent [10], hence, seven predictors form the inputs to the models, and the response (output) variable in this case is the Pressure Drop Coefficient (PDC) as presented in Table I.

A. Pressure Drop Coefficient (PDC)

The pressure drop, ΔP, is often expressed in terms of PDC, which is a dimensionless quantity and a complex non-linear function of cyclone geometrical dimensions (i.e., barrel diameter (*D*), total cyclone height (*H*), vortex finder (*De*), vortex finder length (*S*), inlet height (*a*), inlet width (*b*), height of the cylindrical section (*h*), and the cone-tip diameter (*B*), and operating conditions [13]:

$$\Delta P = PDC \left(\frac{\upsilon_i^2 \rho_g}{2} \right) \tag{1}$$

where υ_i (m/s) is the gas inlet velocity, and ρ_g (kg/m^3) is the gas density. To facilitate a more accurate determination of *PDC*, all eight dimensions of the cyclone are critical as they affect the pressure drop to different extent [13]. Thus, these dimensions are typically characterised by *D* and expressed as seven dimensionless geometric ratios as shown below [13]:

$$PDC = f \left(\frac{De}{D}, \frac{a}{D}, \frac{b}{D}, \frac{S}{D}, \frac{H}{D}, \frac{h}{D}, \frac{B}{D} \right) \tag{2}$$

Since *PDC* is a function of cyclone dimension ratios, as shown in (2), it is not affected by operating conditions such as the inlet gas flow rate, and should remain constant for any cyclone configuration, irrespective of size, provided that the dimension ratios remain the same. Thus, *PDC* can be determined experimentally or theoretically for a particular cyclone, but will be determined more accurately in this work using data-driven techniques that take into account all the geometrical dimensions of the cyclone, and able to generalise well over a wide range of cyclone designs.

TABLE I
INPUT AND OUTPUT VARIABLES FOR THE CYCLONE

	Input							Output
Variables	x_1	x_2	x_3	x_4	x_5	x_6	x_7	z
Spec	De/D	a/D	b/D	S/D	H/D	h/D	B/D	PDC

III. MODEL DEVELOPMENT

A. Fuzzy system (FS)

The heart of a FS, as shown in Fig. 2, is the rule base as it combines all other components to implement the rules [14]. A fuzzy rule base is made up of conditional *IF-THEN* statements:

$$\Re^i : IF\ x_1\ is\ A_1^i\ and\ \cdots\ and\ x_n\ is\ A_n^i,\ THEN\ z\ is\ B^i \quad (3)$$

where $\mathbf{x} = (x_1,\ x_2,\ \cdots,\ x_n)^T \in$ U and z $\in$ V comprise the linguistic input and output variables, respectively, A_j^i and B^i are fuzzy sets defined by membership functions over the input and output universes of discourse, U$_j$ and V, respectively, i = 1, 2, $\cdots$, $\Re$, and $\Re$ is the number of rules in the rule base. The *IF* part of the rule i.e., $x_1\ is\ A_1^i\ and\ \cdots\ and\ x_n\ is\ A_n^i$, is called the antecedent, while the *THEN* part i.e., $y\ is\ B^i$, is called the consequent. This is a case of multi-input single-output system. In depth studies on fuzzy logic and FS can be found elsewhere, e.g., [14][15].

A FS is thus trained to learn a complex input-output mapping from the geometrical measurements of ninety-eight cyclones dataset in [10]. Depending on the type of inference engine, fuzzifier, and defuzzifier used, different combinations of these three modules can result in different FS, however, not all of these combinations make much sense [14]. Here, a type of FS is exploited, which is smooth and continuous, and able to interpolate well between data points and therefore suitable for the current purpose. Hence, a FS with rule base as in (3), product inference engine, singleton fuzzifier, centre average defuzzifier, and Gaussian membership functions, as shown in (4), is a design choice.

$$g(\mathbf{x}|\theta) = \frac{\sum_{i=1}^{\Re} b_i \prod_{j=1}^{n} \exp(-\frac{1}{2}(\frac{x_j - c_j^i}{\sigma_j^i})^2)}{\sum_{i=1}^{\Re} \prod_{j=1}^{n} \exp(-\frac{1}{2}(\frac{x_j - c_j^i}{\sigma_j^i})^2)} \quad (4)$$

$$\theta = [b_1,\ \cdots,\ b_{\Re},\ c_1^1,\ \cdots,\ c_n^1,\ \cdots,\ c_1^{\Re},\ \cdots,\ c_n^{\Re},\ \sigma_1^1,\ \cdots,\ \sigma_n^1,\ \cdots,\ \sigma_1^{\Re},\ \cdots,\ \sigma_n^{\Re}]^T$$

where b_i is the centre of the output fuzzy set for the ith rule, c_j^i is the centre point of the jth input membership function for the ith rule, $\sigma_j^i > 0$ is the width (or spread) of the membership function for the jth input and the ith rule, and θ is the design vector. The FS (4) is designed when the system parameters in θ are determined.

In practice, a FS is built from domain knowledge which is encoded in *IF-THEN* rules. However, when such conscious knowledge is not available, domain input-output data can be collected and used to generate the *IF-THEN* rules and subsequently to construct a type of FS such as in (4). Alternatively, as is the case in this work, a FS structure is first defined, and a suitable algorithm is used to design the system and from which, the rule base can be reconstructed to give an intuitive explanation of the designed FS.

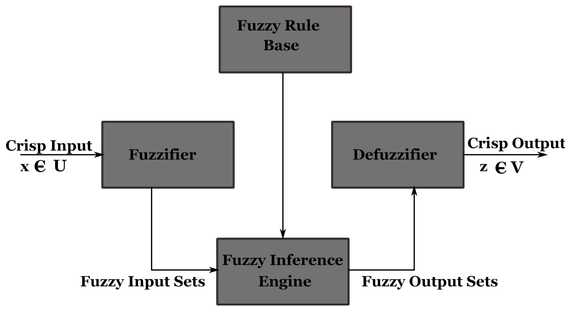

Fig. 2. Schematic diagram of a fuzzy system

B. Gradient descent learning

In order to successfully design (4) and construct a FS that can interpolate well between data points, the gradient descent algorithm is presented to tune all the parameters in θ. The aim of gradient methods is to minimise the squared error, e_l, in (5) for each training data pair, l, by choosing θ.

$$e_l = \frac{1}{2}[g(\mathbf{x}^l|\theta) - z^l]^2 \quad (5)$$

Thus, the gradient descent update law for the Gaussian input centres, c_j^i $(i = 1, 2, \cdots, \Re,\ j = 1, 2, \cdots, n)$, is given as

$$c_{j,t+1}^i = c_{j,t}^i - \lambda \left.\frac{\partial e_l}{\partial c_j^i}\right|_t + \beta(c_{j,t}^i - c_{j,t-1}^i) \quad (6)$$

where the subscripts $t+1$, t and $t-1$ denote the future and past values, $\lambda > 0$ is the learning rate, and $\beta > 0$ is the momentum factor. Chain rule of calculus is applied to $\frac{\partial e_l}{\partial c_j^i}$ at time t, making references to (4) and (5).

$$\frac{\partial e_l}{\partial c_j^i} = e_{l,t} \frac{\partial g(\mathbf{x}^l|\theta_t)}{\partial \mu_i(\mathbf{x}^l, t)} \frac{\partial \mu_i(\mathbf{x}^l, t)}{\partial c_j^i} \quad (7)$$

so that

$$\frac{\partial g(\mathbf{x}^l|\theta_t)}{\partial \mu_i(x^l, t)} = \frac{b_{i,t} - g(\mathbf{x}^l|\theta_t)}{\sum_{i=1}^{\Re} \mu_i(\mathbf{x}^l, t)} \quad (8)$$

and

$$\frac{\partial \mu_i(\mathbf{x}^l, t)}{\partial c_j^i} = \mu_i(\mathbf{x}^l, t) \left(\frac{\mathbf{x}_j^l - c_{j,t}^i}{(\sigma_{j,t}^i)^2} \right) \quad (9)$$

Substituting (8) and (9) into (7) and plugging the result into (6) gives the update equation for the input Gaussian centres, where

$$\mu_i(\mathbf{x}^l, t) = \prod_{j=1}^{n} \exp\left(-\frac{1}{2} \left(\frac{\mathbf{x}_j^l - c_{j,t}^i}{\sigma_{j,t}^i} \right)^2 \right) \quad (10)$$

TABLE II
STATISTICAL SUMMARY OF THE 98 CYCLONES DATASET [10]

	De/D	a/D	b/D	S/D	H/D	h/D	B/D	PDC
Min	0.25	0.113	0.067	0.39	1.158	0.501	0.14	2.3
Mean	0.428	0.630	0.211	0.891	3.283	1.189	0.342	23.268
Standard dev.	0.1104	0.2618	0.0936	0.4289	2.0956	0.6729	0.1498	32.8858
Max	0.667	1.0	0.4	3.052	10.97	3.5	1.0	155.3

Similarly, the gradient update laws for b_i and σ_j^i can be obtained. Here, only the number of rules, $\Re$, in the rule base needs to be set before learning begins, while the system parameters in θ are initialised to some values and tuned to optimum in the course of training. In gradient descent learning, it is necessary to ensure that learning converges before training is finished, thus, longer epochs might be required to achieve this, or the network error is compared to some pre-specified error goal during training and if sufficiently small, training is finished.

C. Multi-Layer Neural Network (MLNN)

The MLNN in Fig. 3 is a three-layer perceptron network. The output, O_i^{m+1}, of any hidden neuron of layer $m+1$ is given by

$$O_i^{m+1} = f_i^{m+1}\left\{\sum_{j=1}^{S^m} w_{i,j}^{m+1} a_j^m + b_i^{m+1}\right\}; \quad m = 0, 1, 2, \cdots, M-1 \qquad (11)$$

where M is the last layer of the network, f_i^{m+1} is the transfer function of the ith neuron of the $(m+1)th$ layer, a_j^m is the output of jth neuron of layer m, and b_i^{m+1} is the bias term associated with the ith neuron of the $(m+1)th$ layer. The designation $w_{i,j}$ is interpreted as the weight connection from input *j* to neuron *i*. In a multi-layer feedforward network, the output of the previous layer becomes the input to the next layer, hence, for the considered two-layer network (excluding the input layer which is just a pass in layer) of Fig. 3, the input vector, $\mathbf{x}$, is vectorised, $\mathbf{x} = [x_1, \cdots, x_7]^T$, and the overall network equation (12) can be constructed, where the input to the last layer is obtained from (11) and the last layer output equation is obtained by setting $m = M-1$ in (11).

$$g = f_1^2\left\{\sum_{j=1}^{S^1} w_{1,j}^2 . f_j^1\left\{\sum_{r=1}^{R} w_{j,r}^1 \mathbf{x}(r) + b_j^1\right\} + b_1^2\right\}. \qquad (12)$$

where R is the dimension of $\mathbf{x}$ which is seven in this case, and $\mathbf{x}(r)$ is the rth element of $\mathbf{x}$.

D. Neural weight learning

The learning of a neural network involves the determination of the non-linear mapping

$$g_t = \Gamma(\mathbf{x}_t, \mathbf{w}) \qquad (13)$$

where the subscript *t* denotes the current time instance, and weight vector, $\mathbf{w}$, is composed of the parameters to be learned. The mapping error, e_t, is the difference between the target output, z_t, and the network output, g_t, i.e., $e_t = z_t - g_t$, and the goal of learning is to find $\mathbf{w}$ which minimises some specified performance criterion. To achieve this, a state-space representation of (13) is first written as

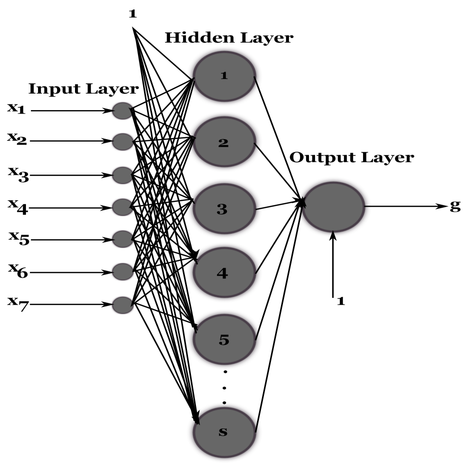

Fig. 3. A three-layer neural network architecture with seven inputs, S hidden neurons and one output neuron

$$\begin{aligned} \mathbf{w_{t+1}} &= \mathbf{w_t} + \mathbf{r_t} \\ d_t &= \Gamma(\mathbf{x}_t, \mathbf{w_t}) + e_t, \end{aligned} \qquad (14)$$

where $\mathbf{w_t}$ is a stationary process with identity state transition matrix driven by process noise $\mathbf{r_t}$, and e_t is the measurement noise. Since $\Gamma(.)$ is a non-linear neural network in this case, it cannot be used directly in the EKF algorithm. Instead, the partial derivatives with respect to network weights are computed and at each time instance t, these derivatives are evaluated at the current predicted state (weight estimate) which are then used in the EKF equations.

1) Recursive EKF algorithm: For EKF implementation, the neural network weight connections including the connections from input to hidden, and hidden to output layers, are coalesced in a single state vector as a linear array. Thus, for the network of Fig. 3, the corresponding Kalman state vector would be written as $\mathbf{w} = [w_{1,1}^1, \cdots, w_{S,R},\ b_1^1, \cdots, b_S^1,\ w_{1,1}^2, \cdots, w_{1,S}^2,\ b_1^2]^T$. Algorithm 1 presents the EKF recursive equations which comprise two steps: prediction and correction. The prediction step forms the predictor for the next observation, and the result is the a priori estimate at time *t*, while the correction step updates the a priori estimate from the prediction with new information arriving at time *t*. This result is the a posteriori estimate.

Algorithm 1 EKF equations

```
Initialisation:
$\hat{\mathbf{w}}_0 = \mathbb{E}[\mathbf{w}]$
$\mathbf{P}_0 = \mathbb{E}[(\mathbf{w} - \hat{\mathbf{w}}_\mathbf{0})(\mathbf{w} - \hat{\mathbf{w}}_\mathbf{0})^T]$
for t = 1 : N$_s$ do                          ▷ Loop over training samples
                                              ▷ Prediction/time update
    $\hat{\mathbf{w}}_{t|t-1} = \hat{\mathbf{w}}_{t-1}$
    $\mathbf{P}_{t|t-1} = \mathbf{P}_{t-1} + \mathbf{Q}$
                                              ▷ Correction/measurement update
    $\mathbf{K}_t = \mathbf{P}_{t|t-1}\mathbf{H}_t(R + \mathbf{H}_t^T\mathbf{P}_{t|t-1}\mathbf{H}_t)^{-1}$
    $\hat{\mathbf{w}}_{t|t} = \hat{\mathbf{w}}_{t|t-1} + \mathbf{K}_t(z_t - \Gamma(\mathbf{x}_t, \hat{\mathbf{w}}_{t|t-1}))$
    $\mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t^T)\mathbf{P}_{t|t-1}$
```

In Algorithm 1, $\mathbf{Q}$ is the process noise covariance, R is the measurement noise variance, $\mathbf{H}_t$ is the first order linearisation of $\Gamma(.)$, evaluated at the current weight estimate, i.e.,$\mathbf{H}_t = \frac{\partial\Gamma(.)}{\partial\mathbf{w}}\Big|_{\mathbf{w}_t}$, $\mathbf{P}$ forms the error covariance of the network weights, and $t|t-1$ denotes the estimate at time t using the available information up to and including the time instance $t-1$.

To begin the recursion, $\mathbf{w}(0)$ is defined by a Gaussian distribution with $\mathcal{N}(\bar{\mathbf{w}}(0), \mathbf{P}(0))$, where $\bar{\mathbf{w}}(0)$ and $\mathbf{P}(0)$ capture any a priori knowledge about the network weights. However, where such knowledge is not available, $\bar{\mathbf{w}}(0)$ can be initialised from a random distribution and $\mathbf{P}(0) = \mu\mathbf{I}$, where μ is a large number, e.g., 10^4.

E. Support Vector Machine (SVM)

SVM, first identified by [16], was originally developed to solve classification problems, but has found application in function approximation/regression as well (see [17]). To configure SVM for regression, the linear ϵ-insensitive loss function (15) is introduced, and the goal here is to find a function $g(x)$ that deviates from the true response by no greater than ϵ for each training instance, and be as flat as possible.

$$L_\epsilon = \begin{cases} 0 & \text{if } |z - g(\mathbf{x})| \leq \epsilon \\ |z - g(\mathbf{x})| - \epsilon & \text{otherwise} \end{cases} \tag{15}$$

Equation (15) means that if the regression error resides within the ϵ-tube, the loss is taken to be zero. If otherwise, the loss is equal to the difference between the regression error and ϵ, where ϵ specifies the desired approximation accuracy. SVM regression problems are easily solved in their Lagrange dual formulation, and to obtain the dual formula for ϵ-insensitive support vector regression, non-negative Lagrange multipliers, α and α^*, are assigned to each training instance, $\mathbf{x}$, and the Karush-Kuhn-Tucker (KKT) first order conditions for optimality are applied. A dual problem results which finds α and α^* that maximises the following quadratic objective function:

$$\begin{aligned} \max_{\alpha,\alpha^*} \quad & -\frac{1}{2}\sum_{i=1}^{N_s}\sum_{j=1}^{N_s}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathbf{x}_i'\mathbf{x}_j - \\ & \epsilon\sum_{i=1}^{N_s}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N_s} z_i(\alpha_i - \alpha_i^*) \\ \text{subject to :} & \\ & \sum_{t=1}^{N_s}(\alpha_i - \alpha_i^*) = 0 \\ & 0 \leq \alpha_i \leq C \\ & 0 \leq \alpha_i^* \leq C \end{aligned} \tag{16}$$

The KKT conditions allow each training point to be classified into support vector types. For non-linear regression problems as is the case in this paper, (16) is kernelised by replacing the linear dot product $\mathbf{x}_i'\mathbf{x}_j$ with a non-linear kernel function, $G(\mathbf{x}_i, \mathbf{x}_j)$, such as the radial basis function. Thus, the approximate prediction function for the SVM regression problem is written as:

$$g(\mathbf{x}) = \sum_{t=1}^{Ns}(\alpha_t - \alpha_t^*)G(\mathbf{x}_t, \mathbf{x}) + b \tag{17}$$

IV. Experimental Methodology, Results and Discussion

A. Cyclone dataset

The cyclone dataset [10] is composed of the measurements of pressure drop for ninety-eight different cyclones gathered from different literature sources. Each cyclone measurement data is composed of seven geometrical dimensionless ratios, and the corresponding PDC. The decision to include a cyclone data in the dataset was based on certain four criteria being met (see [10]). Due to the small dataset available, cross-validation is employed for effective training and generalisation of the intelligent models.

Each column of Table II shows the statistics of each variable. It is observed from the table that there is a large difference in the order of magnitudes between the variables, and *Eu* is noticeably widely dispersed and its mean cannot be said to be representative of the central value considering the range of the data. A more informative and graphical approach to viewing the distribution of the data is with a box and whisker plot. Fig. 4 shows the box plot for each variable, representing information from a five-number summary: minimum value, first quartile, median, third quartile and the maximum value. The red horizontal lines in the boxes denote the median values, the ends of the boxes represent the lower and upper quartiles, and the whiskers are the two lines outside the boxes, extending to the maximum and minimum observations. The data are mostly skewed as the medians cut the boxes into two unequal parts, and the statistics of the variables are significantly different from one another. *S/D*, *h/D* and *Eu* have significant number of outliers as indicated by the '+' symbols. *De/D*, *S/D* and *h/D* have data that are more condensed (closer together) around the larger values (*De/D* and *S/D* have a couple

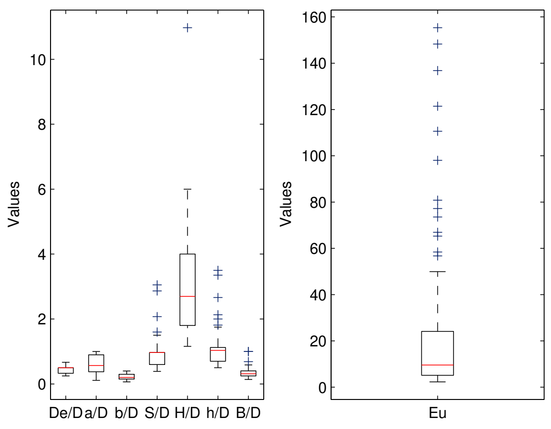

Fig. 4. Statistical distribution of the cyclone dataset

of points with same values as the median/upper quartile since these two coincide), *b/D* and *Eu* are more condensed around the smaller values of the variables. *H/D*, *a/D* and *B/D* are roughly symmetrical about their medians.

B. Data pre-processing

The dataset obtained from [10] and described in the previous section, is the only comprehensive experimental dataset on cyclone that was put together from different literature sources, and is used in the present study for pressure drop prediction through intelligent modelling. To facilitate effective training of the intelligent models, each variable in the dataset is first normalised in the range of [0, 1] according to (18) due to the large difference in the order of magnitude, and subsequently randomly divided into 80% ($\approx$ 78 samples) training and 20% ($\approx$ 20 samples) testing sets.

$$z_n = \frac{z_i - z_{min}}{z_{max} - z_{min}} \tag{18}$$

where z_n is the normalised value of the observed variable, z_i is the actual variable, z_{min} and z_{max} are the minimum and maximum values, respectively, of the observed variable. Both the input and output variables are subjected to the normalisation procedure.

To convert simulation results back to the original scale, de-normalisation is performed using the same normalisation procedure in (18). The processing function essentially becomes an integral part of the models.

C. Hyper-parameters selection/optimisation

Selection of the training parameters (see Table IV) is a critical one. A too small learning rate may result in a long training time and leaning may get stuck, while a value too large achieves faster convergence but may give rise to sub-optimal model parameters. Although an exhaustive search for the learning rate is not considered here, a momentum factor has been introduced to compensate for a sufficiently slow learning rate, and a typical value of 0.01 is sufficiently small

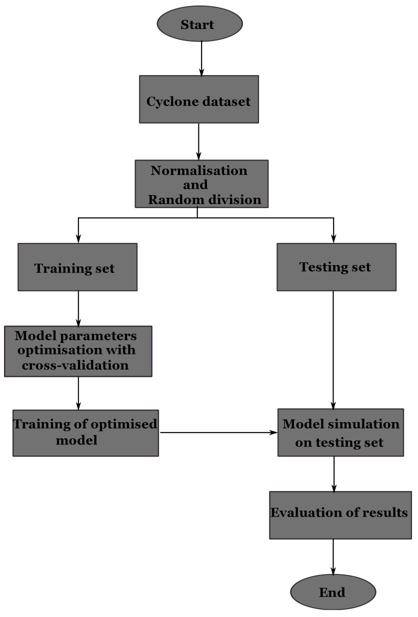

Fig. 5. Procedure used for models development

to this end. More so, the number of rules for the FS is manually tuned with V-fold cross-validation to prevent over-fitting. The value of V has been selected based on the common default V = 10 and considering the small dataset used, this value is considered appropriate for the subsequent analysis. In a V-fold cross-validation experiment, the original sample is randomly divided into V folds, where a single fold is used for independent testing and the other $V - 1$ folds are used to train the model. This process is repeated V times, with each of the V folds being used exactly once for testing. The cross-validation error is the average mean squared error (MSE) on the V testing folds. Thus, $\Re$ = 29 rules is found optimal. Thus, given a total of 29 rules, the first rule in the rule base can be written as *IF* x_1 *is* A_1^1 *and* x_2 *is* A_2^1 *and* $\cdots$ *and* x_7 *is* A_7^1 *THEN* z *is* B^1, where $A_1^1 \cdots A_7^1$ and B^1 are linguistic values defined by Gaussian functions whose parameters $c_1^1 \cdots c_7^1$, $\mu_1^1 \cdots \mu_7^1$ and b_1, are available in θ. Similarly, the remaining 28 rules are constructed and the rule base is recovered. Linguistic inputs can be partitioned into a number of linguistic values (e.g., small, large) depending on the application and domain knowledge. However, in this work, partitioning is not given importance as the number of rules is optimally determined to achieve sufficient regression accuracy.

Furthermore, the number of hidden neurons, N_h, influences the performance of neural networks. Several rules of thumb exist in the literature to fix the size of the hidden layer with each resulting in different values on the same application (an indeterminate situation), and none of which guarantees optimality in any one application. Thus, the search for hidden neurons is an optimisation problem. Applying the same procedure, N_h = 14 with hyperbolic tangent neurons. The search for optimal SVM parameters employs a grid search technique with 10-fold cross-validation for the regularisation parameter, C, the radial basis kernel scale factor, K_s, and ϵ. Optimal search for these parameters is performed in the range of C = [0.001, 1000], K_s = [0.001, 1000] and ϵ = [1e-04, 8.24]. And the optimal values found are 1000, 2.154 and 3.82e-03, respectively.

D. Model training

The optimal models, based on the parameters obtained previously, are trained on the entire dataset used for cross-validation, i.e., all the V-folds are put together to design the final prediction models and the abilities of the trained models to generalise on unseen future data are determined on the remaining 20% testing set, held out for independent testing purposes. Fig. 5 sets out the training/testing procedure used in this work. Prior to training, the choice of initial parameters is critical to the success of gradient descent algorithms. The closer the initial values are to the optimum, the higher the chance of converging on the optimum. Thus, FS parameters are initialised with the first $\Re$ training instances, i.e., $c_j^i(0) = x_{j,0}^i$, $b_i(0) = z_{i,0}$, and $\sigma_j^i(0)$ = [max$(x_{j,0}^i : i = 1, 2, \cdots, \Re)$ − min$(x_{j,0}^i : i = 1, 2, ..., \Re)$]/$\Re$, where $i = 1, 2, \cdots \Re$, $j = 1, 2, \cdots, n$. The neural network weight, $\mathbf{w(0)}$, on the other hand is drawn from a random distribution in the [0, 1] range and $P(0) = 1e4\mathbf{I}$. Ideally, $\mathbf{w(0)} = 0$ is appropriate, random initialisation did produce much better response in this application.

To confirm the generalisation ability and robustness of the AI models, which are needful for determining the prediction accuracies of the models, Fig. 6 and Table III present the results on the testing samples. Generally, gradient descent-based learning is easy to converge on a shallow local minimum. However, by integrating cross-validation, the optimised FS and BPNN models are improved and hence generalise well on test data, with correlation coefficient (R) values of 0.9970 and 0.9953, respectively. Despite the improvements, the EKF-based learning results in a slightly better prediction performance over the gradient descent both in terms of MSE and R, and thus deserves popularisation. SVM models on the other hand are not considered universal approximators, but have yielded results comparable to fuzzy and neural model architectures, where the SVM parameters have also been optimally determined via cross-validation for fair comparison. Experiments also reveal that the SVM performance is comparably low when the kernel is polynomial (MSE = 1.3e-03, R = 98.52%) or linear (MSE = 1.4e-02, R = 82.1%). Thus, the radial basis kernel is found suitable for the SVM in this work with R = 0.9975. On the overall, slight improvement could make the difference in practice, and the EKF-MLNN provides the superior performance on the prediction of PDC to this end. That said, the choice of learning algorithm matters in training an AI model, and is worth the test with different learning algorithm, especially when high precision is of paramount importance.

To ensure that the learned weights converge, Figs 7 and 8 show the weight learning curves for the EKF and gradient descent training, respectively. EKF stabilises at the 143^{rd} epoch and after which, learning only proceeds very slowly, while gradient descent requires much longer epochs to converge to the result obtained. The back-propagation used here is based on the momentum gradient descent algorithm, and comparing EKF to back-propagation, the following are observed: i) BPNN takes longer epochs (about 10 times longer than that of EKF-MLNN) to converge, ii) 21 hidden neurons are optimal for BPNN to achieve the mapping accuracy as compared to 14 needed by EKF-MLNN, iii) EKF, however, requires more parameters to be tuned/initialised to get excellent result. However, if higher prediction accuracy is desired, the experiment herein has shown that EKF-trained MLNN could reach an R value of 0.9982, and thus the model of choice. But trade-offs might be necessary in practice.

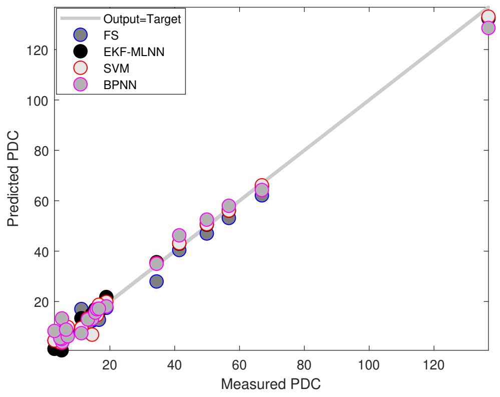

Fig. 6. Comparison of the AI models for PDC prediction

TABLE III
MODELS' TESTING PERFORMANCE

	FS	**EKF-MLNN**	**SVM**	**BPNN**
R	0.9970	0.9982	0.9975	0.9953
MSE	3.97e-04	1.62e-04	2.22e-04	4.89e-04

E. Comparison with theoretical, semi-empirical and multi-regression models

The EKF-MLNN model is compared to conventional cyclone pressure drop models developed by [4][5][6]. It is clear from Fig. 9 that EKF-MLNN clearly outperforms the conventional theoretical, semi-empirical and multi-regression models, and the smallest MSE in Table V is about 45 times larger than that of MLNN. The model of Shephered & Lapple [4] is surprisingly better than others in terms of MSE and R values because this model does not take into account the effect of all cyclone dimensions. However, this may be attributed to the nature of the data used for testing. It is noteworthy

TABLE IV
MODELS' PARAMETERS AND SIMULATION VALUES

FS	**MLNN**	**SVM**
λ = 0.01	R = 1.1	C = 1000
β = 0.72	Q =1e-03$\mathbf{I}$	K_s = 2.154
$\Re$ = 29	N_h = 14	ϵ = 3.82e-03

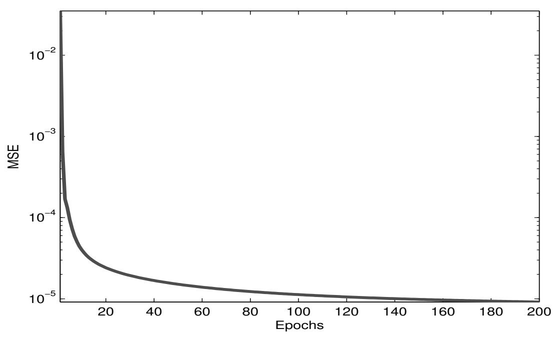

Fig. 7. Convergence of the EKF weights learning

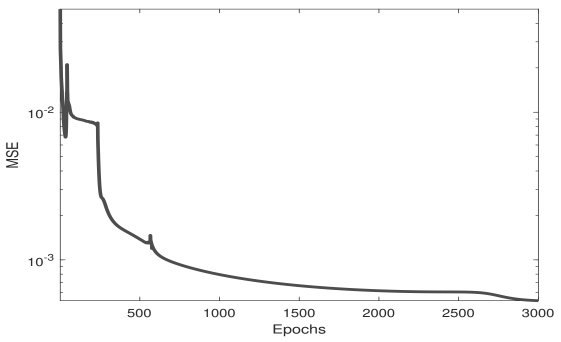

Fig. 8. Convergence of the gradient descent weights learning

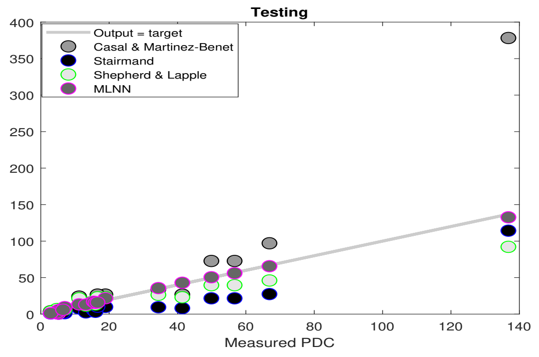

Fig. 9. Comparison of MLNN with conventional cyclone pressure drop models

TABLE V
PERFORMANCE OF CONVENTIONAL MODELS TO EKF-MLNN

	MSE	**R**
Casal and Martinez-Benet[6]	1.23e-01	0.9441
Shepherd and Lapple[4]	7.3e-03	0.9788
Stairmand[5]	1.4e-02	0.9399
EKF-MLNN	1.62e-04	0.9982

that none of the conventional models considered matches the accuracy of any of the intelligent methods.

V. CONCLUSION

This article proposes three methods to improve the pressure drop prediction of cyclone separators. Models hyper-parameters are first determined manually and by grid search (for SVM) in a V-fold cross-validation setting to prevent over-fitting. The obtained optimal models are subsequently trained and validated on previously unseen data and results show that EKF learning of MLNN performed best, and the R value could reach 0.9982, thus deserving popularisation in the future. Furthermore, the EKF and the back-propagation algorithm are compared and EKF is shown to be better in terms of early convergence, smaller hidden layer size and prediction accuracy. In comparison with conventional pressure drop models, this work shows that all AI models are able to achieve a maximum reduction error of about 99% on testing samples, and thus able to predict pressure drop more accurately over a wide range of cyclone designs. Notable limitations of the developed AI models are that adequate performance is only guaranteed in the range of dimension ratios for which they have been trained, and EKF is only guaranteed to achieve first-order (Taylor series) accuracy. It is envisaged that future work will use the best performing pressure drop model developed in this work and an efficiency model, in a multi-objective optimisation framework, to design an optimal cyclone geometrical configuration that achieves a suitable compromise between two conflicting objectives: minimum pressure drop and maximum efficiency.

REFERENCES

[1] J. Dirgo and D. Leith, "Cyclone collection efficiency: comparison of experimental results with theoretical predictions," *Aerosol Science and Technology*, vol. 4, no. 4, pp. 401–415, 1985.
[2] L. Wang, M. D. Buser, C. B. Parnell, and B. W. Shaw, "Effect of air density on cyclone performance and system design," *Transactions of the ASAE*, vol. 46, no. 4, p. 1193, 2003.
[3] B. Zhao, "Modeling pressure drop coefficient for cyclone separators: a support vector machine approach," *Chemical Engineering Science*, vol. 64, no. 19, pp. 4131–4136, 2009.
[4] C. Shepherd and C. Lapple, "Flow pattern and pressure drop in cyclone dust collectors cyclone without intel vane," *Industrial & Engineering Chemistry*, vol. 32, no. 9, pp. 1246–1248, 1940.
[5] C. Stairmand, "Pressure drop in cyclone separators," *Industrial and Engineering Chemistry*, vol. 16, no. B, pp. 409–411, 1949.
[6] J. Casal, "A better way to calculate cyclone pressure drop," *Chem. Eng.*, pp. 90–99, 1983.
[7] W. Griffiths and F. Boysan, "Computational fluid dynamics (cfd) and empirical modelling of the performance of a number of cyclone samplers," *Journal of Aerosol Science*, vol. 27, no. 2, pp. 281–304, 1996.
[8] W. Barth, "Design and layout of the cyclone separator on the basis of new investigations," *Brenn. Warme Kraft*, vol. 8, no. 1, p. 9, 1956.
[9] C. J. Stairmand, "The design and performance of cyclone separators," *Trans. Instn. Chem. Engrs.*, vol. 29, pp. 356–383, 1951.
[10] G. Ramachandran, D. Leith, J. Dirgo, and H. Feldman, "Cyclone optimization based on a new empirical model for pressure drop," *Aerosol Science and Technology*, vol. 15, no. 2, pp. 135–148, 1991.
[11] P. K. Swamee, N. Aggarwal, and K. Bhobhiya, "Optimum design of cyclone separator," *AIChE journal*, vol. 55, no. 9, pp. 2279–2283, 2009.
[12] J. Gimbun, T. Chuah, A. Fakhru'l-Razi, and T. S. Choong, "The influence of temperature and inlet velocity on cyclone pressure drop: a cfd study," *Chemical Engineering and Processing: Process Intensification*, vol. 44, no. 1, pp. 7–12, 2005.
[13] B. Zhao and Y. Su, "Artificial neural network-based modeling of pressure drop coefficient for cyclone separators," *chemical engineering research and design*, vol. 88, no. 5-6, pp. 606–613, 2010.
[14] L.-X. Wang, *A course in fuzzy systems*. Prentice-Hall press, USA, 1999.
[15] L. A. Zadeh, *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*, vol. 6. World Scientific, 1996.
[16] V. N. Vapnik, *The nature of statistical learning theory*. Heidelberg, DE: Springer science Verlag, 1995.
[17] V. N. Vapnik, "An overview of statistical learning theory," *IEEE transaction on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.