Analysis and Modeling of Multivariate Nuclear Power Plant data using VARMA and LSTM

Kunal Taneja
Department of ECE , University of Waterloo
https://github.com/kunal-taneja/SYDE675

December 2, 2019

Abstract

Accurate modeling and simulation of a power plant are important factors in strategic planning and maintenance of the system. Several non-linearities and multivariable couplings are associated with real world plants. Therefore, it becomes almost impossible to model the system using conventional mathematical equations. Statistical models such as ARIMA, ARMA are potential solutions but their linear nature cannot very well fit a system with non-linear relationships between its parameters. Therefore, in such cases, VARMA (Vectored ARMA) model becomes the key candidate for multivariate time series forecasting. Recently, deep learning methods such as Artificial Neural Networks have been extensively applied for time series forecasting. In this project, we thoroughly discuss the implementation of a key architecture of neural networks, Long Short-Term Neural Networks (LSTMs). LSTM has the ability to capture nonlinear relationships in a dynamic system using its memory connections. This further helps them to overcome the issue of back-propagated error decay through memory blocks. The objective of this study is to examine the feasibility of LSTMs and VARMA models in modelling of Nuclear Power Plant and predicting the system parameters such as Reactor Temperature based on past information.

Index terms - Multivariable couplings, Artificial Neural Networks, LSTM, ARIMA, ARMA, VARMA, Temperature prediction

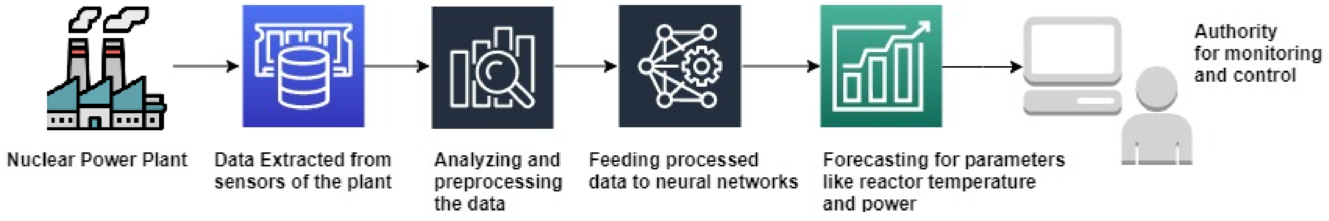
1 Introduction

1.1 Motivation

The demand for primary energy and electricity is projected to increase by 50 percent and 70 percent, respectively[1]. The International Energy Agency (IEA) encourages research on renewable energy related technologies for a sustainable future due to the rising threat of depletion of fossil fuels. Nuclear energy is a brightly emerging energy source in the current days. Even though generation of electric power from nuclear energy is a topic of controversy, if handled and harnessed correctly, it can serve humanity for many centuries to come with abundant and clean energy.

In many counties, energy markets are divided in next-day and intra-day market sessions. There is a continuous pressure on the power plants to supply a required amount of energy. According to the energy market's supply demand conditions and predictions, the plant management can know if the predicted requirement could be met. Through this knowledge the plant authorities can make decisions regarding energy purchases. The sooner is commitment is made, lower is the price. As the energy market follows a bid system, larger the error in power prediction, larger is the cost the plant authorities have to pay to honor the contracts. Therefore, forecasting of energy availability remains a top priority in the industry[2]. Moreover, due to the errors in forecasting the power requirement, the nuclear reactor's output needs to be modulated which in place reduces the overall efficiency of the whole system. Physical models used for simulation are based on the physics of the processes and components of the power plant. As the plant ages, the uncertainty of a physical model to correctly represent the plant increases over time. ANN models can be trained on the data collected overtime from the sensors to take care of the degradation of the plant. Moreover, by comparing the predictions of models over time, the plant degradation can be assessed[3]. However, ANNs being a black box model cannot empirically provide us with mathematical equation for the process. VARMA being a special case of ARMA, ARIMA models for multivariate modeling represent the process empirically. But being a parametric model it has an overhead of estimating the order of the autoregressive and moving average terms.

Our study contributes in implementation of LSTM and VARMA models for short term and long term forecasting. LSTM belongs to the family of neural networks, which have shown great performance in applications such as text analytics, time series analysis, computer vision and many more[4]. The objective of the models is to predict the reactor temperature of the nuclear reactor given appropriate past information. Developed models are tested on real data of the same plant to check the model viability of on-line applications.


Nuclear Power Plant
Data Extracted from sensors of the plant
Analyzing and preprocessing the data
Feeding processed data to neural networks
Forecasting for parameters like reactor temperature and power
Authority for monitoring and control

1.2 Problem Statement

Nuclear power plants utilize the nuclear fission energy of uranium isotope to heat up the water in the boilers to yield super-heated steam which is later used to run the steam turbines to produce electric energy. The whole process of generating this energy is very critical, with the main parameters being reactor temperature, boiler pressure, feed water temperature, steam line pressure, pre-heater temperature and turbine vibrations. The reactor temperature measurement is a critical parameter for both safe and economic operation of the plant. The data being analyzed is obtained from a power plant whose reactor's temperature has been increasing over time and is one of the several parameters preventing the unit to produce optimal power. Due to the fluctuations in the reactor temperature, the current set point of reactor temperature is set below the optimal value to tackle any uncertainty in temperature measurement. A truly representative model of the reactor would help to reduce this uncertainty so that the reactor can be operated safely at its true optimal limit. According to the process design even a minor control in the fluctuation of the reactor temperature can lead to significant increase in the power output of the turbine. We, hereby, aim to model the behaviour of the process using LSTM networks and VARMA models. This will authorise the station to raise the RIHT (reactor temperature) values closer to peak values of the safety limit, ensuring better power output as well as plant's revenue.

2 Methodology

2.1 Vectored Auto Regressive Moving Average (VARMA) Models

Vector auto-regressive moving average models (VÄRMA) are used in various multivariate forecasting problems, particularly in time series analysis to reveal the cross-correlations between various features of the time stamped data [8]. It is extensively used in applications of finance and econometric.
Suppose the data under study has n related time series components such that $z_t = z_{1t}, z_{2t}, ..., z_{nt}$. A Vectored Auto-regressive Process (VAR) is defined as:

$$z_t = A_1 z_{t-1} + ... + A_p z_{t-p} + a_t \tag{1}$$

where the $A_i's$ (i=1,...,p) are $n \times n$ coefficient matrices corresponding to the auto-regressive terms and a_t is the n-dimensional error term. All the components of the time series are interdependent. Values of one component depend on the lags of itself and other components. Therefore, we represent this relation as:

$$z_{1t} = A_{1(11)} z_{1(t-1)} + ... + A_{1(1p)} z_{p(t-1)} + a_{1(t-1)} \tag{2}$$

Similar equations can be written for the other components corresponding to other lag relations. VAR models are used in one time step ahead forecasting or longer forecasting by using one step ahead method recursively [10]. For complex datasets in order to accurately represent the behaviour of time series signal, large order (p value) for VAR may be needed. Hence a large number of parameters might be required for true description of the data. Moreover, in this case VAR may suffer from uncertainty in parameter estimation[10].

Therefore, a larger model class of vectored auto-regressive moving average (VARMA) is considered for modeling of complex datasets. A VARMA process is defined as:

$$A_0 z_t = A_1 z_{t-1} + ... + A_p z_{t-p} + M_0 a_t + M_1 a_{t-1} + ... + M_q a_{t-q} \tag{3}$$

where A_i's (i=1,...,p) and M_j's (j=1,...,q) are the $n \times n$ auto-regressive and moving average parameters respectively.

VARMA models were developed for stationary and non-seasonal data originally and work well for such kind of data. In order to check for the stationarity of multivariate data, Johansen's co-integration test is performed and the eigenvalues obtained in its result are checked to comment on the stationarity of the data.

2.2 Artificial Neural Networks

In this section, we will review artificial neural networks (ANNs) along with their application in time series analysis. ANNs have revolutionized a wide range of machine learning tasks from domains such as natural language processing to computer vision by ability to generalize/learn a given data distribution through examples. The foundation of ANNs was introduced early by [11]; however, due to lack of computational resources, the area remained under-explored. With recent development of computational technology and extended exposure to GPUs, the ANNs have emerged showcasing their true potential. It was initially proposed to mimic a human brain. So, a major part of its design is inspired by the structure of human brain cells. Fig. 1 shows different components of an ANN where each node represents a *neuron* and an arrow represents a connection from the output of one neuron to the other. We now discuss the two primary components of any ANN.

2.2.1 Neurons

Trying to mimic human brain cells, ANNs borrowed the concept of neurons and how they processed an input and produced an output. More precisely, input signals to a neuron are passed through an *activation function* for normalization and to add robustness to the model. This means that a small change in input would result in a small variation in the output. These neurons are responsible to maintain a *state* which when combined together are expected to represent feature vectors for a given task.

2.2.2 Connection and Weights

Each arrow in the Fig. 1 shows a *connection*, responsible for passing signals amongst connected neurons. Each connection is assigned a weight that controls the importance of the particular connection. It is important to note that a neuron can have multiple incoming and outgoing connections, each with a different weight. When training an ANN for a specific task, we adopt a learning regime where we update the weights of each neuron and calculate the error/loss. This error is distributed across the neurons using an optimization algorithm such as *back-propagation*[12], which provides feedback for adjusting the weights of the neurons and guides the model towards convergence. We now discuss the different types of ANNs and how they are used in time series analysis.

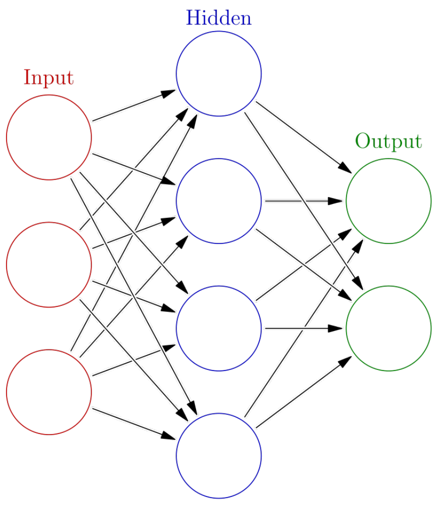


Figure 1: ANN architecture.
Source: `https://en.wikipedia.org/wiki/Artificial_neural_network`

In Fig 1, we can observe the three different types of neurons:

1. red *input* neurons which accept the input data.
2. green *output* neurons that make the feature vector when combined together.

3. blue *hidden* neurons that model the transformation of signal from input and output neurons.

The arrows represent a *connection* and each of them have a weight that controls the contribution/importance of signal from a neuron.

2.3 Multi-Layer Perceptron (MLP)

A multilayer perceptron (MLP), also known as a feedforward network, is composed of multiple layers of preceptrons, each coupled with an activation function. The two most common choices of non-linear activation functions are sigmoids such as *tanh* and rectifier linear unit (ReLU) [13].
The training procedure of an MLP involves updating connection weights after each data sample is processed such that the overall error accumulated in one pass of all the data samples, in other words, in one *epoch*, is minimized. The final output of an MLP can be expressed as:

$$y_t = \alpha_0 + \sum_{j=1}^{m} \alpha_j \mathcal{H}(\beta_{0j} + \sum_{i=1}^{n} w_{ij} x_{ti}) + \varepsilon_t, \forall t, i \in \{0, 1, \ldots, n\}, j \in \{0, 1, \ldots, m\} \quad (4)$$

Here, x_{ti} are the n inputs and y_t is the final output of the network. m and n represent the number of input and hidden neurons, respectively. α_0 and β_{0j} are the bias terms, and α_j and w_{ij} are the different connection weights. $\mathcal{H}$ denotes an activation function such as *tanh* or a *ReLU* and ε_t denotes a random shock parameter. As we can see, an MLP essentially performs a non-linear transformation from inputs (past observations of the time series) and outputs (future value). A simplified form of MLP is presented in Eq.5

$$y_t = f(y_{t-1}, y_{t-2}, \ldots, y_{t_n}, \theta) + \varepsilon_t \quad (5)$$

Here, θ is the set of parameters and f is the transformation learned by the model by adjusting the connection weights.

2.4 Recurrent Neural Networks (RNNs)

A Recurrent Neural Network (RNN) [12] is a special type of neural network designed for efficiently processing sequential data. This makes them a suitable candidate for a task such as time-series analysis. The "recurrent" part depicts the fact that these networks apply the same set of operations on all the nodes to model the temporal relationship of data samples.

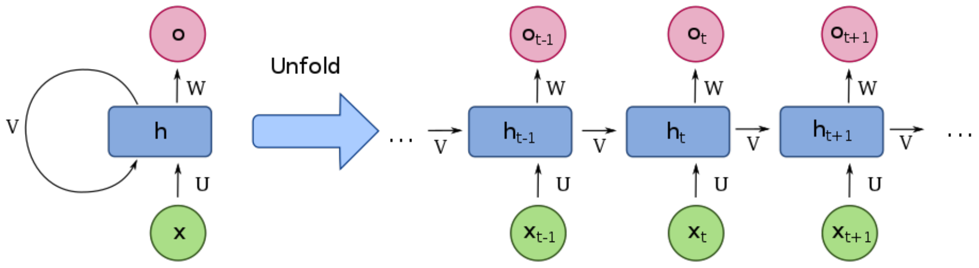


Figure 2: Unfolding in recurrent neural networks
Source: `https://en.wikipedia.org/wiki/Recurrent_neural_network`

2.4.1 “Unfolding” in RNNs

RNNs work by “unfolding” the computational graph and using information of the past as feedback. Fig. 2 shows the unfolding mechanism in an RNN. For an input x the network processes the information by incorporating it into the state h that is passed forward in time. The loop v denotes lag of one time-step. The right part of Fig 2 is an unfolded view of the network, where each node is associated with one single time step. We can see how input at a given time step takes into account, information from the previous time steps. It is also important to note that RNNs are usually “deeper” than vanilla MLPs and due their feedback mechanism, are proven to model the input-output relationship in a dataset very well.

2.5 Long Short Term Memory (LSTM)

The Long Short Term Memory model belongs to a family of recurrent neural netwokrs (RNN). RNNs are conceptually different from feed forward neural network (FFNN) as shown in the images, as there are no feedback loops in the FFNNs. The concept of FFNNs was conceived in 1940s in order to understand and mimic the working of human brain. So, RNNs can be viewed an advancement over FFNNs and LSTMs can be seen as an advancement over RNNs.
RNNs have a closer relationship to the functioning of a human neuron than FFNNs as RNNs lean in a progressive fashion rather than a randomized order. The architecture of RNN allows hidden units to share parameters across time indexes, which effectively helps in building a memory of linked sequences. However, the vanishing gradient problem prohibits the model from learning from deep sequences. Even our brains have a difficult time recalling memories and events from the past[5]. LSTM addresses the problem of vanishing gradient by using self-connected gates in the hidden units.These gates allow the LSTM units to read, write and remove information from memory. Which is equivalent to remembering certain amount

of information and forgetting the irrelevant information[6].

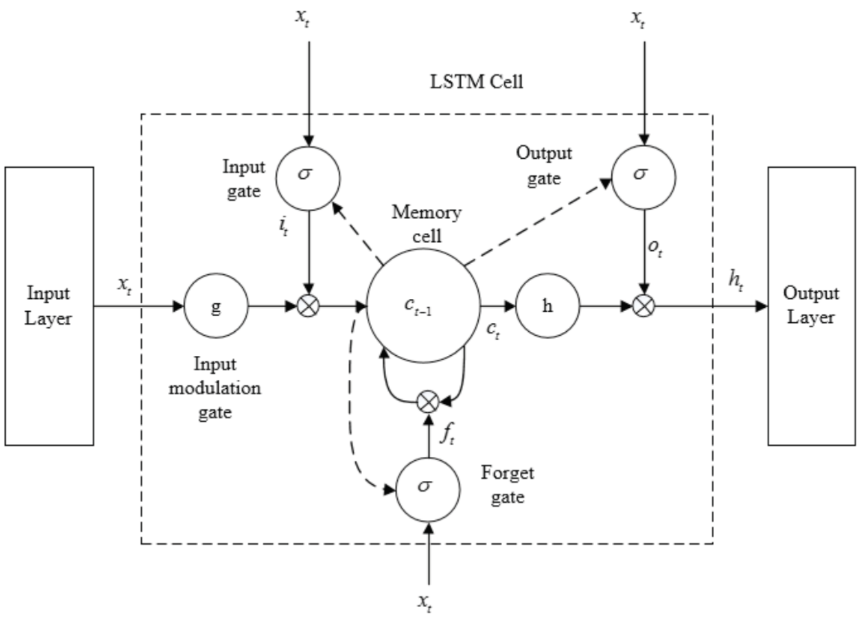


Figure 3: Structure of an LSTM cell[8].

- **Functioning:** The typical structure of a LSTM cell has been shown in Fig 3. A typical LSTM has 4 gates: input gate, input modulation gate, forget gate and output gate. Input gate takes a new input point from the outside and processes the incoming data. Memory gate takes input from the last output of the cell. Forget gate decides when to forget the output results and provides a time lag to select an input sequence. Output gate generates the output of the cell after processing the inputs from other gates.

  Notations:
  1. input time series: $X = (x_1, x_2, ..., x_n)$

  2. hidden state of memory cells: $H = (h_1, h_2, ..., h_n)$

  3. output time series: $Y = (y_1, y_2, ..., y_n)$

$$h_t = H(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{6}$$
$$p_t = H(W_{hy}y_{t-1} + b_y) \tag{7}$$

  Equation (6) describes the computation done by LSTM where W denotes the weight matrix and b denotes bias matrix. Hidden state of LSTM memory cell is computed as:

$$i_t = \sigma(W_{ix}x_t + W_{hh}h_{t-1} + W_{ic}c_{t-1} + b_i) \tag{8}$$

$$f_t = \sigma(W_{fx}x_t + W_{hh}h_{t-1} + W_{fc}c_{t-1} + b_f) \tag{9}$$

$$c_t = f_t * c_{t-1} + i_t * g(W_{cx}x_t + W_{hh}h_{t-1} + W_{cc}c_{t-1} + b_c) \tag{10}$$

$$o_t = \sigma(W_{ox}x_t + W_{hh}h_{t-1} + W_{oc}c_{t-1} + b_o) \tag{11}$$

$$h_t = o_t * h(c_t) \tag{12}$$

here σ stands for hyperbolic tangent function which defined as:

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{13}$$

Objective function for our model is root mean squared error (rmse) which is a measure of how varied or spread out the prediction is from the ground truth. Lower rmse values contribute towards better predictions. Higher the rmse, more deviated is the prediction from the actual trend.

$$rmse = \sqrt{\frac{1}{n}\Sigma(y_t - p_t)^2} \tag{14}$$

In order to avoid the problem of local minimum we use Adam optimizer instead of stochastic gradient descent optimizer for back propagation through time (BPTT). This ensures minimum training errors with adaptive learning rates[8].

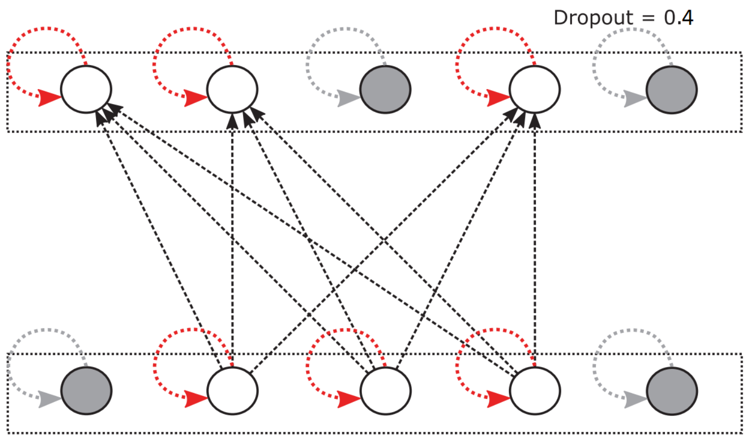


Figure 4: Example of dropout layer application[2].

- **Regularization**: Neural networks are generally prone to over-fitting. As the network learns and adjusts weight, sometimes it not only retains the knowledge about the structural patterns in the data but also the eccentricities and noise specific to the training data. To remove over-fitting, we use the common practice of dropout regularization [6]. In dropout regularization, a random set of cell units are blocked and do not communicate with other cells in that iteration. Removing connections reduces the number of parameters to be estimated while training and the overall complexity of the network. Thus, dropout helps preventing over-fitting. Dropout is user defined parameter can be set by using default values or cross-validation. Network that includes a dropout can be consisted as an ensemble, with each model for an epoch of unblocked units. For instance, in the Fig 4, the model has a 40 percent dropout rate, that means for every epoch 40 percent of the total units do not contribute towards the model training. Hence, while predicting, we implicitly average over the predictions for all the models corresponding to each epoch, which is equivalent to the concept of ensemble learning.

3 Exploratory Data Analysis

The data is collected from the four sensors installed in reactor unit A and reactor unit B of the power plant namely: RIHT sensor, Boiler Pressure sensor, Steam Line Pressure sensor,Feed Water Temperature sensor.

Therefore, every parameter in the system is associated with a sensor. The data is collected over a span of almost 9 years, sampled daily. Raw data retrieved from the sensors was prone to noise and outliers, so, to ensure the integrity of the dataset several filtering techniques were employed to denoise the data. The clean data retrieved after denoising is our primary area of study which is analysed in the following sections.

In this section, we understand the nature of the data and therefore analyse the properties of the data. Time-series data is decomposed into several components to look for any trend or seasonality present in it. Stationarity tests such as Augmented Dickey Fuller (ADF) test and Johansen's co-integration test are performed over the data to check whether time series is stationary or not.

3.1 ADF test for Stationarity

Augmented Dickey Fuller Test is a unit root test for stationarity of a time series data. ADF test is an augmented version of Dickey fuller test which is used for more complex time series data. The test has following hypothesis:

- **Null Hypothesis H_o:** There exists a unit root for the time series data.

- **Alternate Hypothesis H_a:** stationarity or trend-stationarity based on which version of test is used.

The test outputs a test statistic value which is a negative number. The more negative the number is, higher is the degree of rejection of null hypothesis[14]. ADF is implemented in various software packages. Python uses **statsmodels.tsa.stattools** package to implement ADF test.Following is the python code for implementation of ADF test:

```
from statsmodels.tsa.stattools import adfuller
def stationarity_test(series):
    output=adfuller(series)
    print('ADF Stastistic:',output[0])
    print('p-value:',output[1])
    pvalue=output[1]
    for key,value in output[4].items():
         if output[0]>value:
            print("The time series is non stationary")
            break
         else:
            print("The time series is stationary")
            break;
    print('Critical values:')
    for key,value in output[4].items():
        print(key, value)
```

The above mentioned code returns test statistic value along with critical values. If test statistic value is greater than any of the critical values, we reject the null hypothesis and state time series is not stationary. Similar thing is signified by the p-value also returned by the code. If p value is greater than 0.05 (confidence interval) then we reject the null hypothesis with enough evidence.

3.2 Johansen's Co-integration test

Johansen's test allows us to verify whether two or more time signals have cointegrating relationship or not. It is well suited for multivariate forecasting techniques such as VARMA. It works on Vector Error Correcetion Model (VECM) and the autoregressive parameters of the multivariate model are decomposed and eigenvectors and eigenvalues are analysed. In order for the process to be stationary, the eigenvalues of inverted autoregressive operator must be less than 1.

We use coint_johansen package from the **statsmodels.tsa.vector_ar.vecm** library in python and apply .eig() mehod to it in order to get the eigenvalues.

```
from statsmodels.tsa.vector_ar.vecm import coint_johansen
values = data.values
coint_johansen(values,-1,1).eig
```

3.3 Analysing daily dataset extracted from Unit B of the power plant

The time series plot below describes the measurements of each parameter extracted from their respective sensors over a period of almost nine years. RIHT and Feed Water Temperature are measure in °C, Main Steam Line Pressure and Boiler Pressure are measured in kPa (Kilo Pascals). Sampling frequency for the below mentioned plot is one observation per day. The data plotted in the figures is obtained after handling the outliers and the missing values. The data corresponding to the plant's power shutdown has also been removed.

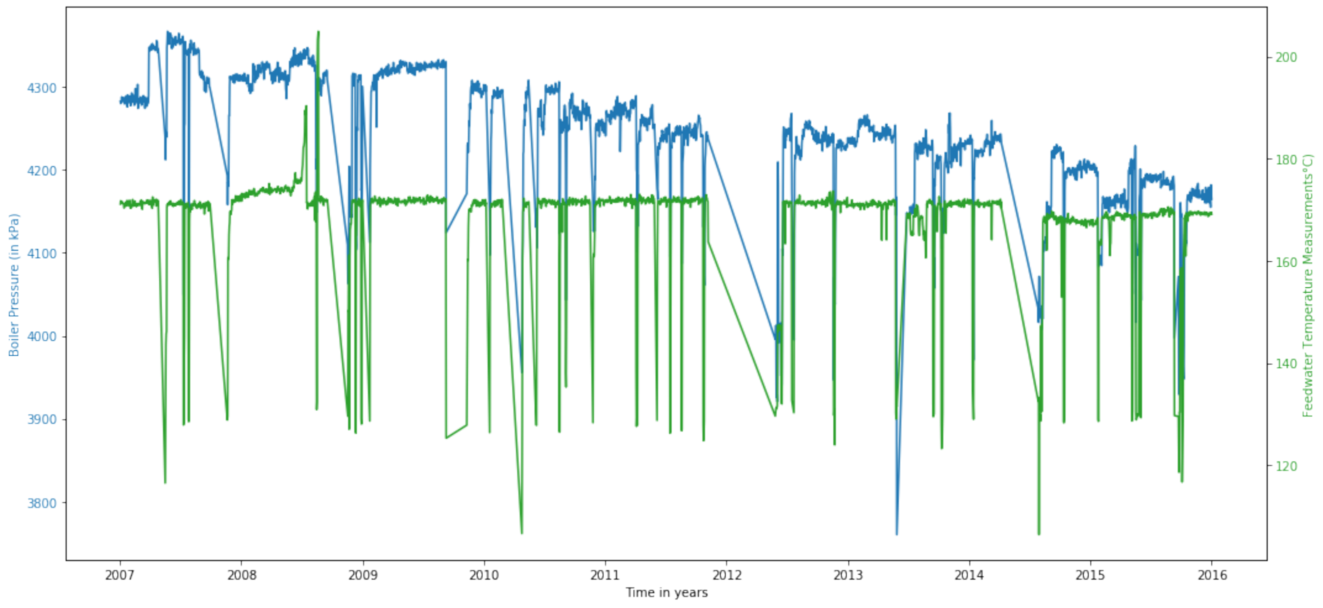


Figure 5: Plot for Boiler Pressure data vs Feedwater Temperature data from Unit B.

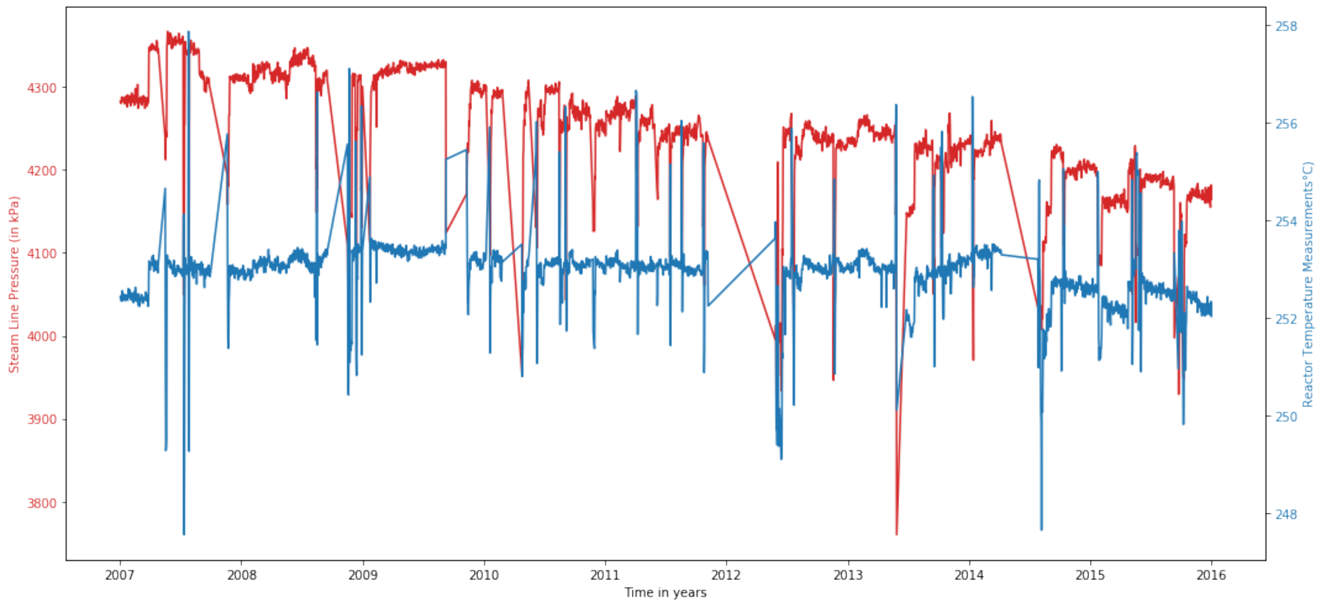


Figure 6: Plot for RIHT data vs Steam Line Pressure data from Unit B.

On observing Fig. 5 and Fig. 6, we find certain parameters have similar variation patterns with respect to time. For instance, there exists correlation between RIHT and Boiler Pressure and also between RIHT and Main Stream Line Pressure. However, this correlation cannot be modelled by linear methods. We test the stationarity of each feature using the ADF test. Table 1, 2, 3, 4 outline the ADF test carried on all 4 features. p-values and ADF statistic values of all the test suggest stationarity of the data.

Table 5 illustrates the results of johansens's test. The eigenvalues of the inverse of autoregressive operator are all less than 1 which further signifies the stationary multivariate data.

Augmented Dickey Fuller Test for Stationarity				
ADF-Statistic	**p-value**	**Critical value at 1%**	**Critical value at 5%**	**Critical value at 10%**
-7.701	0.000	-3.433	-2.863	-2.567

Table 1: ADF test for stationarity of RIHT data (Unit B)

Augmented Dickey Fuller Test for Stationarity				
ADF-Statistic	**p-value**	**Critical value at 1%**	**Critical value at 5%**	**Critical value at 10%**
-3.993	0.001	-3.433	-2.863	-2.567

Table 2: ADF test for stationarity of Boiler Pressure data (Unit B)

Augmented Dickey Fuller Test for Stationarity				
ADF-Statistic	**p-value**	**Critical value at 1%**	**Critical value at 5%**	**Critical value at 10%**
-15.155	0.000	-3.433	-2.863	-2.567

Table 3: ADF test for stationarity of FeedWater data (Unit B)

Augmented Dickey Fuller Test for Stationarity				
ADF-Statistic	**p-value**	**Critical value at 1%**	**Critical value at 5%**	**Critical value at 10%**
-3.880	0.002	-3.433	-2.863	-2.567

Table 4: ADF test for stationarity of Steam Line data (Unit B)

Next, we study the nature of parameters in whose forecasts we are interested in, i.e. RIHT. Fig. 7 represents the data for Reactor Temperature from 2007-2015. On visualising the data, no particular trend can be observed. We can

Johansen's Cointegration Test for Stationarity			
EigenValue 1	**EigenValue 2**	**EigenValue 3**	**EigenValue 4**
1.84239210 e-01	3.02510284 e-02	2.41131810 e-02	2.30883029 e-08

Table 5: Johansen's test for stationarity of multivariate power plant data (Unit B)

check for any trends or seasonal components present in this time series using **seasonal_decompose** function of **statsmodels.tsa.seasonal** package available in Python. Fig 8 shows the components of RIHT data. There is no general trend found in the data. Also, no proper seasonality is captured in the decomposition.

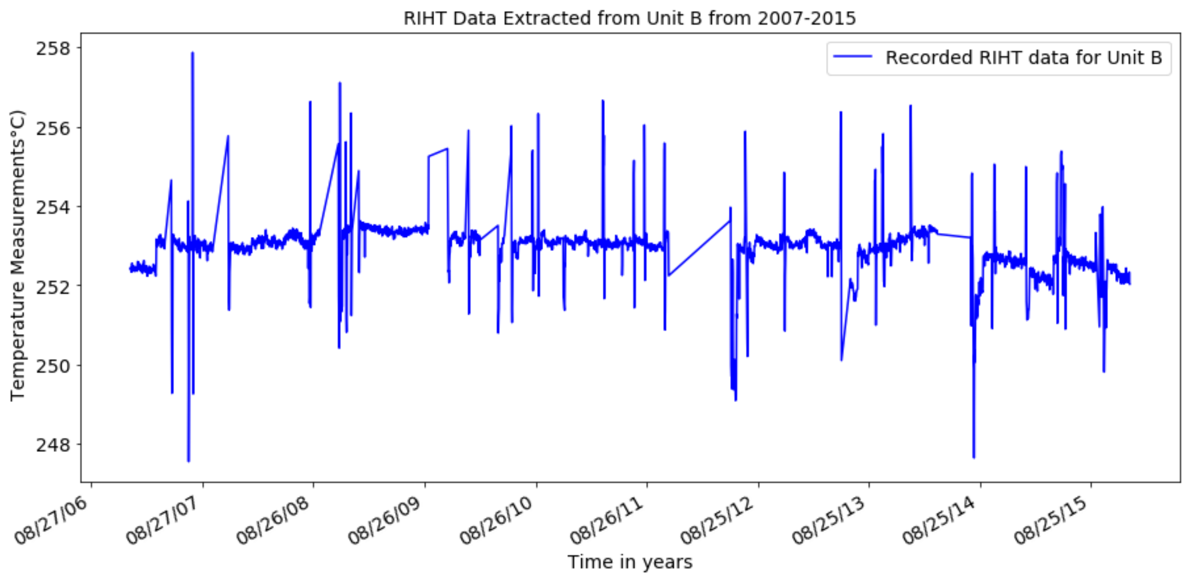


Figure 7: Plot for RIHT data from Unit B.

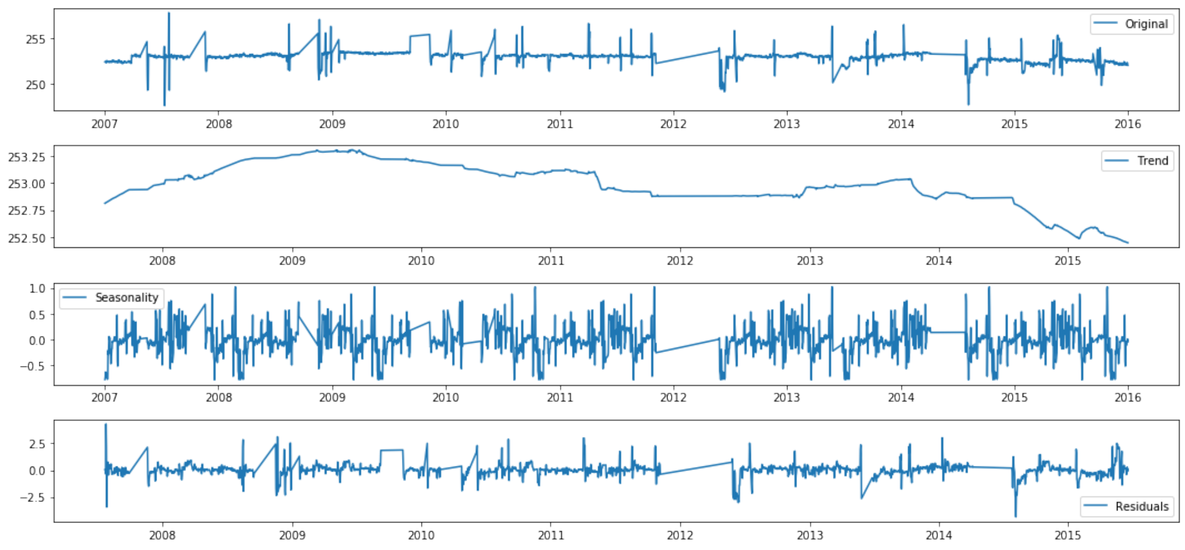
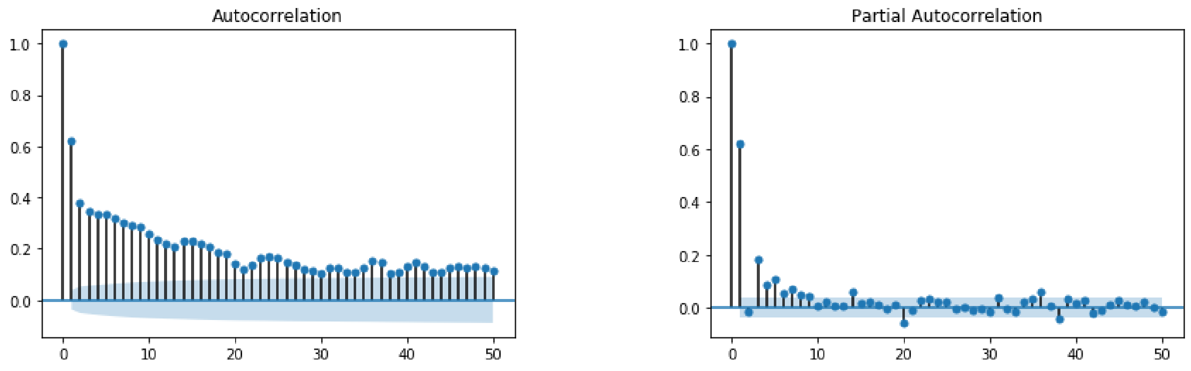


Figure 8: Plot for components of RIHT data after decomposition.

4 Implementation and Results

4.1 Using VARMA for RIHT prediction

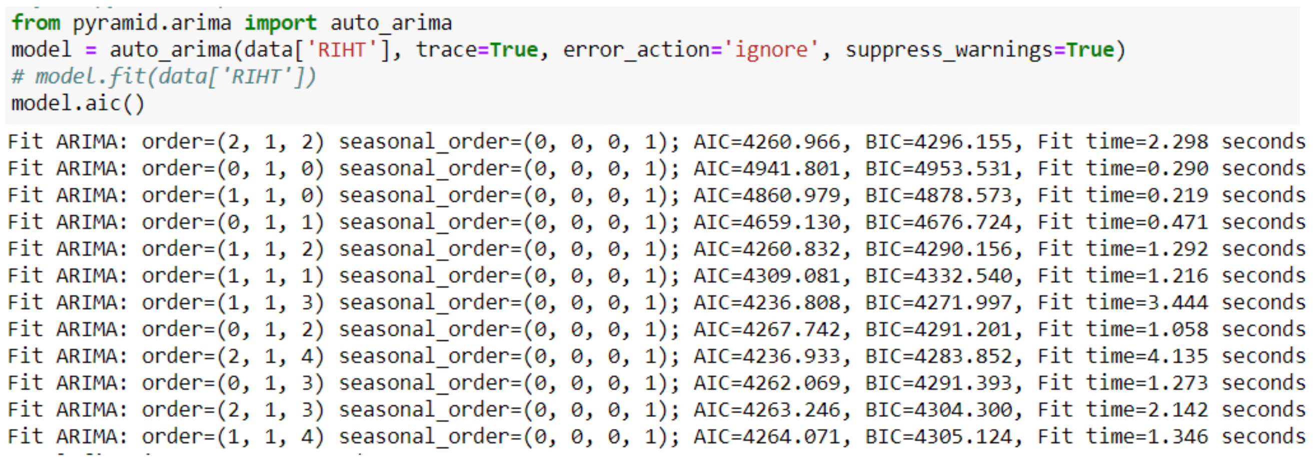
Fitting a VARMA model requires proper selection of autoregressive and moving average order, i.e. value of p and q. So, similar to ARMA/ARIMA modeling we use ACF and PACF plots of the system parameters to obtain the optimal values of p and q. Fig. 9 represents the ACF and PACF plots for the RIHT data obtained from Unit B.



(a) Auto-correlation plot for RIHT data (b) Partial Auto-correlation plot for RIHT data

Figure 9: ACF and PACF plots

We cannot choose values of p and q confidently just by looking these plots. Therefore, we use auto.arima method implemented in **pyramid-arima** in python to optimally select the values of p and q. Following are the results obtained by using auto.arima method on RIHT data.



```
from pyramid.arima import auto_arima
model = auto_arima(data['RIHT'], trace=True, error_action='ignore', suppress_warnings=True)
# model.fit(data['RIHT'])
model.aic()
```

```
Fit ARIMA: order=(2, 1, 2) seasonal_order=(0, 0, 0, 1); AIC=4260.966, BIC=4296.155, Fit time=2.298 seconds
Fit ARIMA: order=(0, 1, 0) seasonal_order=(0, 0, 0, 1); AIC=4941.801, BIC=4953.531, Fit time=0.290 seconds
Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 0, 0, 1); AIC=4860.979, BIC=4878.573, Fit time=0.219 seconds
Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 0, 0, 1); AIC=4659.130, BIC=4676.724, Fit time=0.471 seconds
Fit ARIMA: order=(1, 1, 2) seasonal_order=(0, 0, 0, 1); AIC=4260.832, BIC=4290.156, Fit time=1.292 seconds
Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 0, 0, 1); AIC=4309.081, BIC=4332.540, Fit time=1.216 seconds
Fit ARIMA: order=(1, 1, 3) seasonal_order=(0, 0, 0, 1); AIC=4236.808, BIC=4271.997, Fit time=3.444 seconds
Fit ARIMA: order=(0, 1, 2) seasonal_order=(0, 0, 0, 1); AIC=4267.742, BIC=4291.201, Fit time=1.058 seconds
Fit ARIMA: order=(2, 1, 4) seasonal_order=(0, 0, 0, 1); AIC=4236.933, BIC=4283.852, Fit time=4.135 seconds
Fit ARIMA: order=(0, 1, 3) seasonal_order=(0, 0, 0, 1); AIC=4262.069, BIC=4291.393, Fit time=1.273 seconds
Fit ARIMA: order=(2, 1, 3) seasonal_order=(0, 0, 0, 1); AIC=4263.246, BIC=4304.300, Fit time=2.142 seconds
Fit ARIMA: order=(1, 1, 4) seasonal_order=(0, 0, 0, 1); AIC=4264.071, BIC=4305.124, Fit time=1.346 seconds
```

Figure 10: Selecting ARIMA of order (2,1,2) as the optimal model for RIHT data.

Now fit of ARIMA (2,1,2) is tested over RIHT data. A good model is one which after fitting returns residuals that correspond to white noise process, i.e. have no correlation. Following equation describes the model which is fit on the RIHT data:

$$(1 - 0.0224B - 0.0949B^2)z_t = (1 - 0.4192B - 0.4630B^2)a_t - 0.0001 \qquad (15)$$

where z_t correspond to the response series (RIHT) and a_t is the white noise. Fig. 11 represents the fit of ARIMA(2,1,2) on the RIHT data. Fig. 12 shows the residual plot along with its density graph. We observe that it is normally distributed around mean zero. Also by looking at ACF and PACF plots in Fig. 13 we conclude that residuals have no correlation among themselves. Hence, ARIMA (2,1,2) is a good model for modeling RIHT data.

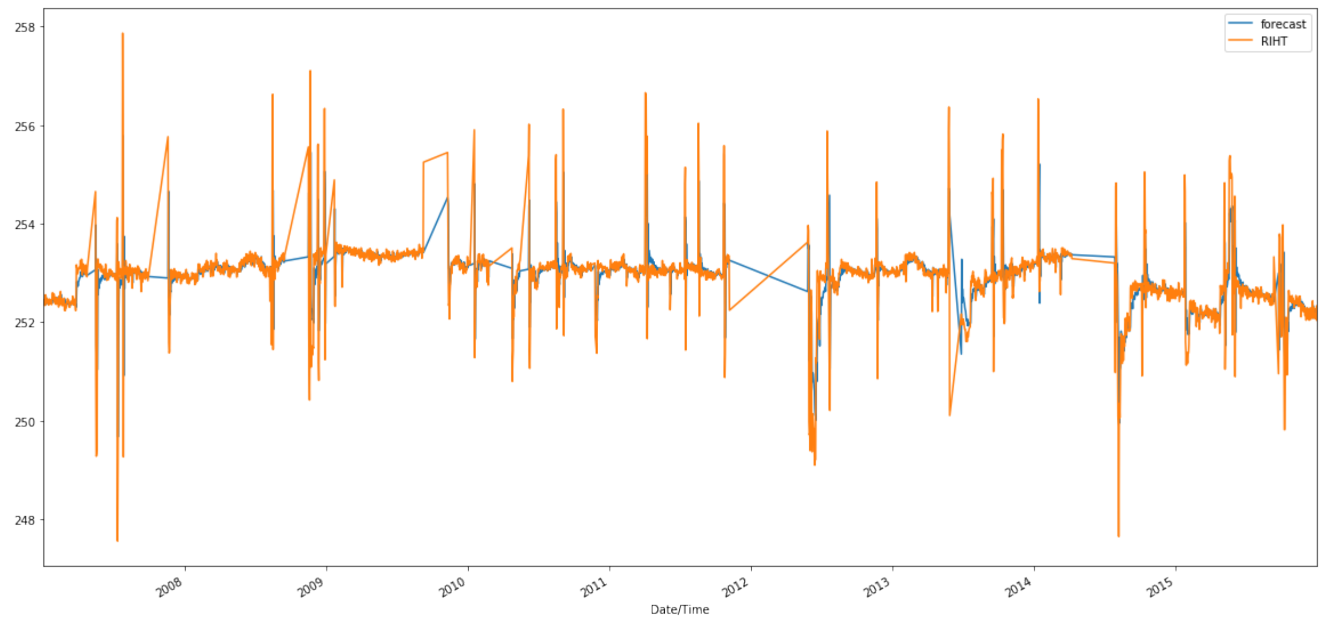


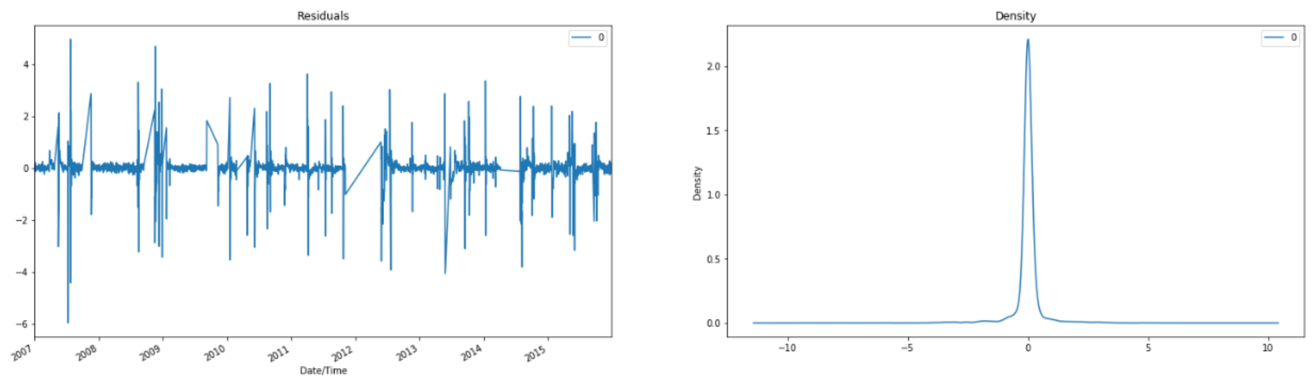
Figure 11: Performance of ARIMA (2,1,2) on RIHT data.

Even after running auto.arima method on all other time signals we got (2,1,2) as the common optimal order for ARIMA modeling on respective time series. Therefore, order (2,1,2) is selected for VARMA modeling of the multivariate plant data.

Using VARMA(2,1,2) for Short term prediction of RIHT: VARMA of order (2,1,2) is selected as per discussion in previous sections. Now it is used to make short term predcitions of RIHT data based on training over past information. Model is trained on multivariate data from Jan 2007-Nov 2015 and is used to make predictions for RIHT in Dec 2015.

Using VARMA(2,1,2) for Long term prediction of RIHT: Model is trained on multivariate data from Jan 2007-Dec 2014 and is used to make predictions for RIHT from Jan2015 - Dec 2015.

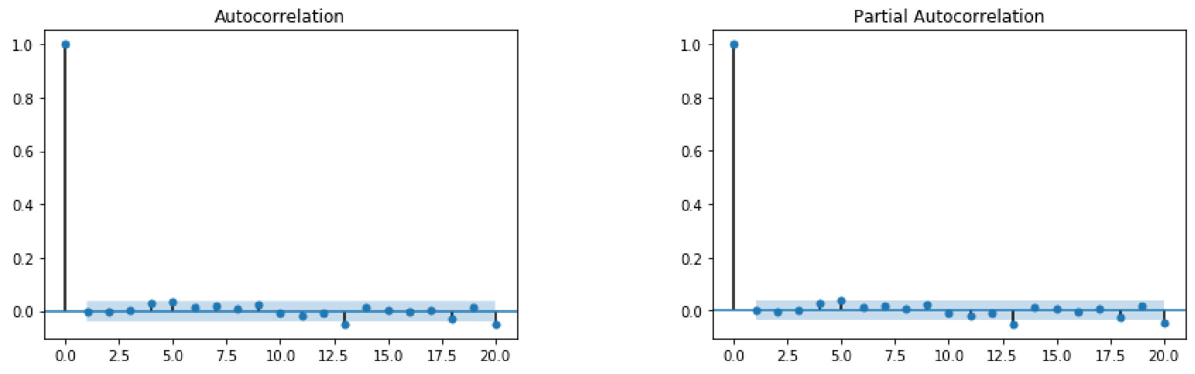
```
residuals = pd.DataFrame(model_fit.resid)
fig, ax = plt.subplots(1,2)
residuals.plot(title="Residuals", ax=ax[0])
## kde=kernel density estimation
residuals.plot(kind='kde', title='Density', ax=ax[1],figsize=(25,7))
plt.show()
```



```
residuals.mean()
```

```
0    0.000019
```

Figure 12: Plot representing residual data along with its density function.



(a) Auto-correlation plot for residuals (b) Partial Auto-correlation plot for residuals

Figure 13: ACF and PACF plots

Performance metrics for VARMAX(2,1,2) model			
Duration	**RMSE**	**Maximum Error**	**Minimum Error**
Short Term	0.271	0.472	0.0101
Long Term	0.527	2.549	0.0003

Table 6: Performance Metrics for RIHT forecasting using VARMAX

Fig. 14 and Fig. 15 display the results for short term and long term prediction of RIHT respectively. Table 6 outlines the performance metrics of the VARMA(2,1,2) used for both short term and long term predictions.

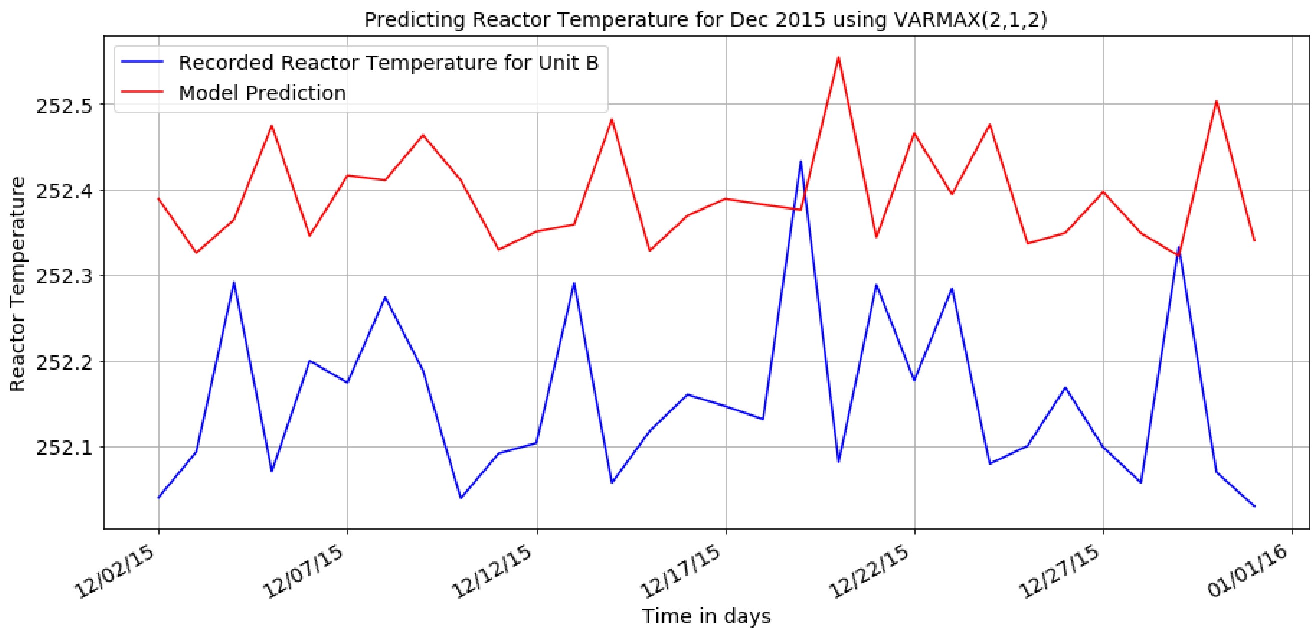


Figure 14: Predicting RIHT for Dec 2015.

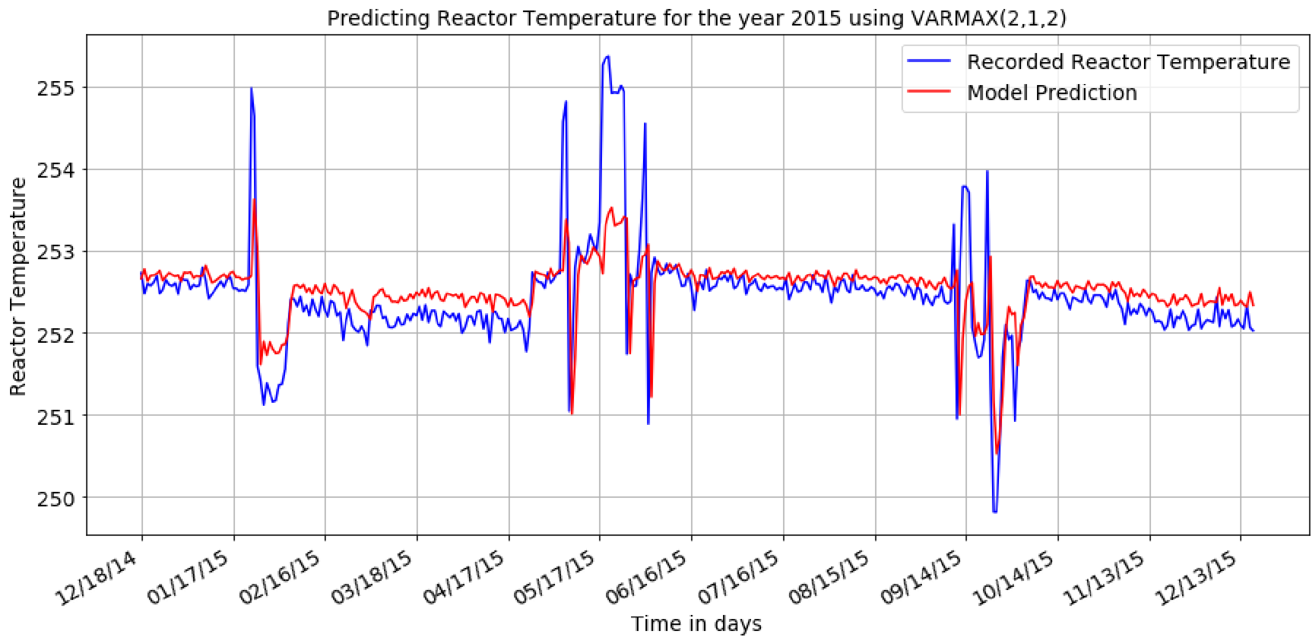
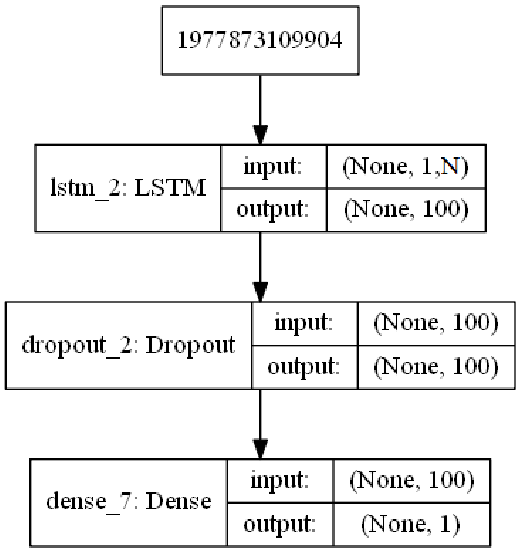


Figure 15: Predicting RIHT for year 2015.

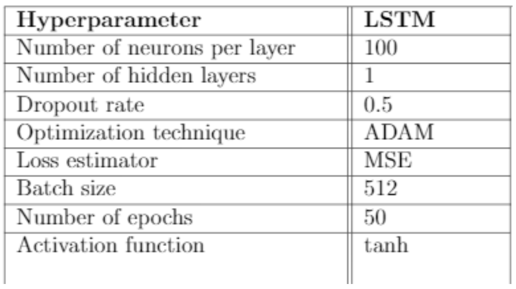
VARMA(2,1,2) is able to make predictions as it catches the trend of the actual values. The only drawback of this method is high training time along with selection of optimal order of p and q.

4.2 Using LSTM for RIHT prediction

The proposed LSTM model is applied to the data collected from four sensors installed in the power plant. It constitutes one hidden layer with 100 neurons and one dense layer with one neuron contributing towards the prediction of reactor temperature. Hence, 100 memory units in the spatial axis of the LSTM network are modelled. As the sampling frequency of our data is 1 time unit therefore, time lag is set as an integer with multiples of 1 in the temporal axis. Fig. 16 represents the architecture of the LSTM network along with the Hyperparameters selected for training over multivariate data.



(a) LSTM network architecture



Hyperparameter	LSTM
Number of neurons per layer	100
Number of hidden layers	1
Dropout rate	0.5
Optimization technique	ADAM
Loss estimator	MSE
Batch size	512
Number of epochs	50
Activation function	tanh

(b) Hyperparameters selected for training

Figure 16: LSTM Network

As the LSTM model is initially trained over the known values of input and output parameters, the choice of these parameters is not dependent on cause and effect relations like in physical modelling. While applying neural network models the selection of parameters is more dependent on the objective and availability of reliable data. For predicting RIHT at time-step t we use values of reactor temperature, feed water temperature, boiler pressure and steam line pressure at the (t-1)th time instance. Fig. 17 shows the proposed prediction model.

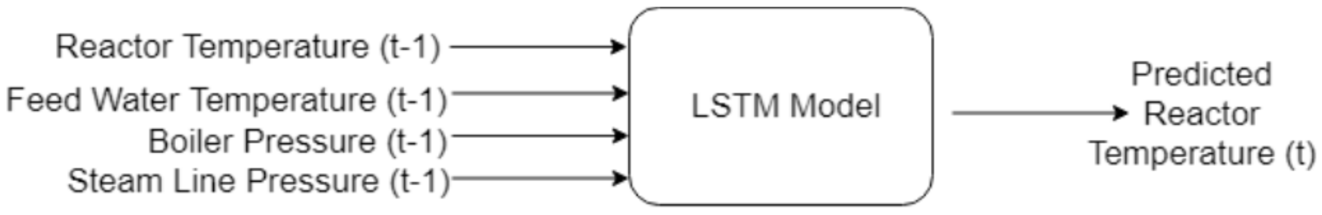

Reactor Temperature (t-1)
Feed Water Temperature (t-1)
Boiler Pressure (t-1)
Steam Line Pressure (t-1)
LSTM Model
Predicted Reactor Temperature (t)

Figure 17: LSTM model for RIHT prediction.

4.2.1 Using LSTM for short and long term RIHT prediction

Short term prediction: Training LSTM network for multivariate data from Jan 2007 - Nov 2015 and predicting RIHT for Dec 2015.

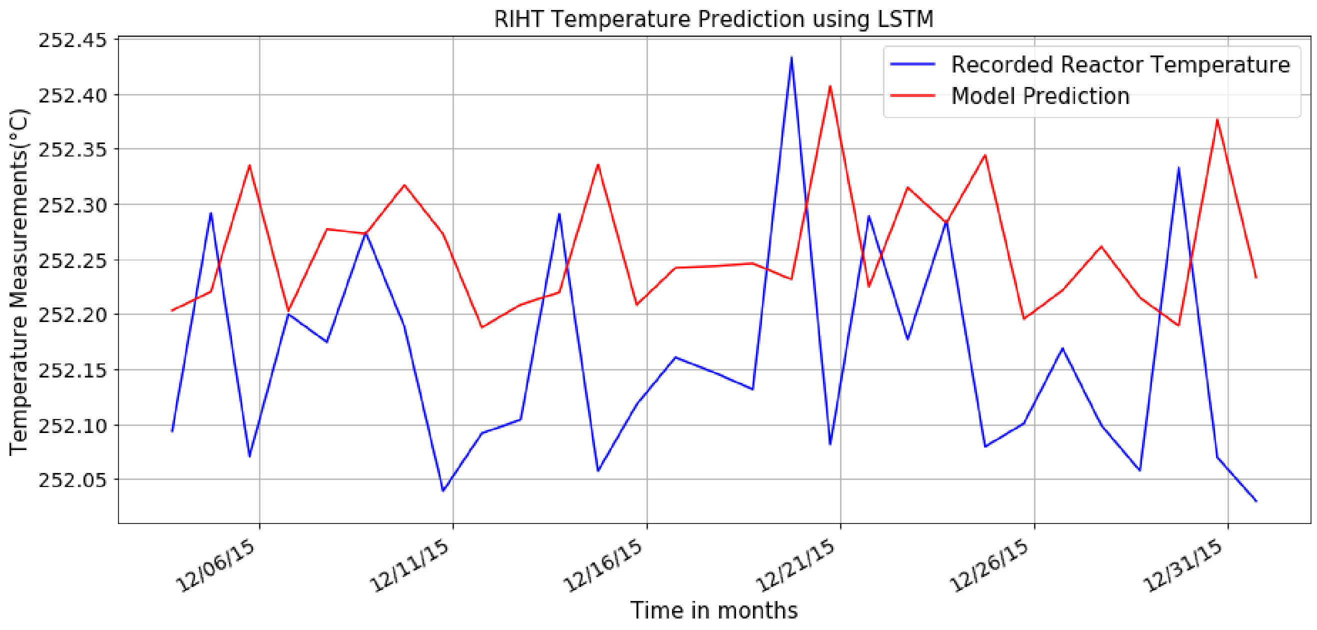


Figure 18: Using LSTM for short term prediction

Long Term prediction: Training LSTM network for multivariate data from Jan 2007 - Dec 2014 and predicting RIHT for year 2015.

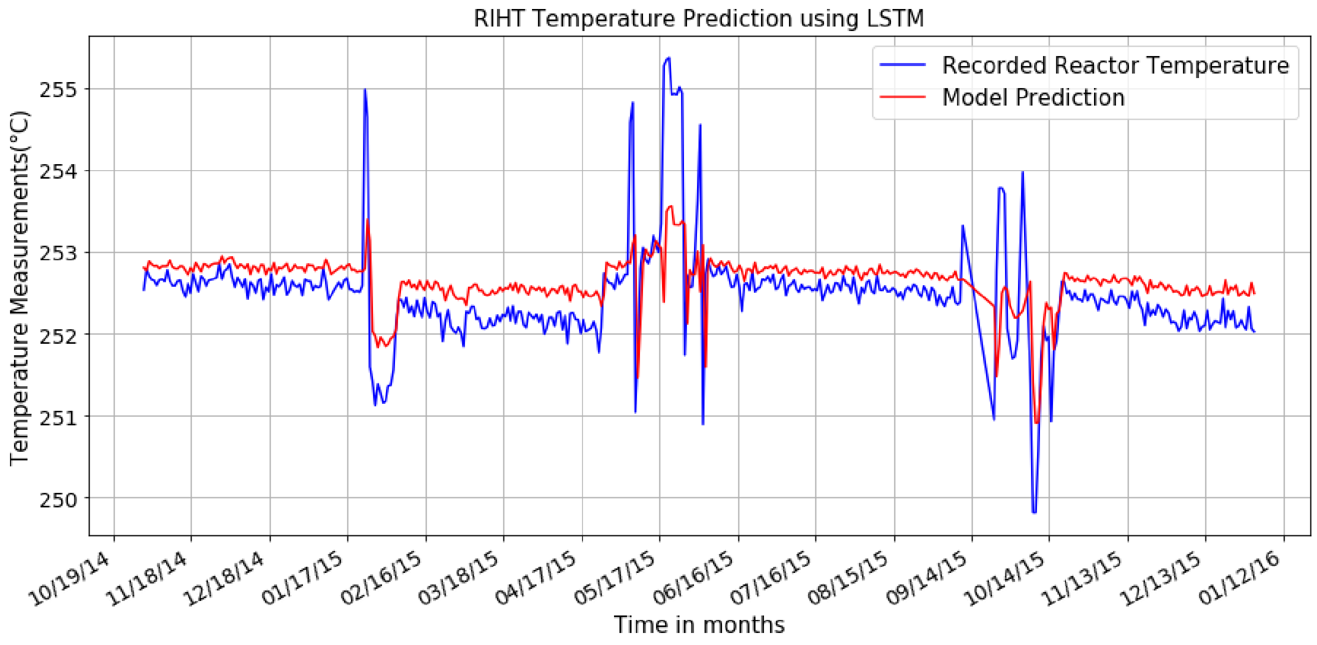


Figure 19: Using LSTM for long term predictio

Fig. 18 and Fig. 19 show the output of LSTM model in prediction of RIHT for short and long term prediction respectively. We observe that LSTM model

Performance metrics for LSTM model			
Duration	**RMSE**	**Maximum Error**	**Minimum Error**
Short Term	0.163	0.325	0.0014
Long Term	0.528	2.886	0.0003

Table 7: Performance Metrics for RIHT forecasting using LSTM

performs better than VARMA in case of short term prediction. This can also be seen in the results of performance metrics reported in Table 7 when compared to Table 6. However, in case of long term the quality of predictions of LSTM is similar to that of VARMA.

5 Conclusion

The main purpose of this project was to analyse the real data obtained from various sensors installed at the nuclear power plant and then forecast the values for Reactor Temperature(RIHT) using VARMA and LSTM. Development of accurate physical models for nuclear power plant can be difficult due to the complexity of the system. Physical models tend to lose their accuracy over time due to plant's degradation. Stochastic models like ARMA, ARIMA are candidate solutions but they only work well on univariate data. Therefore, we make application of VARMA models in order to deal with multivariate forecasting. With VARMA we get a proper empirical model for the data with several parameters their coefficients. Therefore, given this model we can simply extract values of target varibales by feeding the values of input parameters. However, it suffers from the problem of large training periods as well as proper selection of order (p and q) values for the AR and MA operators.
Deep learning methods such as LSTM are applied on the multivariate data and their feasibility in prediction of RIHT is checked. The performance of LSTM networks is compared with VARMA models for both short and long term predictions. LSTM perform better than VARMA model when it comes to short term predictions whereas it produces similar results compared to VARMA for long term predictions. Training time for LSTM network is quite low and being non-parametric model it doesn't require selection of any parameters prior to training process. However, being a black box model LSTM do not provide true representation of any kind of empirical model in contrast to VARMA method.

Considering real world application of forecasting in power plants, smaller training times and better quality of short term and long term predictions make LSTM networks as a proper fit for this multivariate forecasting problem.

References

[1] Duffy, Rogers, and Ayompe, “Renewable energy and energy efficiency assessment of projects and policies,” Billy-Blackwell, 2015.

[2] Srivastava, Shikhar, and Lessmann. ”A comparative study of LSTM neural networks in forecasting day-ahead global horizontal irradiance with satellite data.” Solar Energy 162 (2018): 232-247.

[3] Smrekar, Jure, et al. ”Development of artificial neural network model for a coal-fired boiler using real plant data.” Energy 34.2 (2009): 144-152.

[4] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. ”Deep learning.” nature 521.7553 (2015): 436.

[5] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. ”Learning long-term dependencies with gradient descent is difficult.” IEEE transactions on neural networks 5.2 (1994): 157-166.

[6] Hochreiter, Sepp, et al. ”Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.” (2001).

[7] Szegedy, Christian, et al. ”Going deeper with convolutions.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[8] Fu, Rui, Zuo Zhang, and Li Li. ”Using LSTM and GRU neural network methods for traffic flow prediction.” 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC). IEEE, 2016.

[9] Simionescu, Mihaela. ”The use of VARMA models in forecasting macroeconomic indicators.” Economics and Sociology 6.2 (2013): 94.

[10] Lütkepohl, Helmut. ”Forecasting with VARMA models.” Handbook of economic forecasting 1 (2006): 287-325.

[11] McCulloch, Warren S., and Walter Pitts. ”A logical calculus of the ideas immanent in nervous activity.” The bulletin of mathematical biophysics 5.4 (1943): 115-133.

[12] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. ”Learning representations by back-propagating errors.” nature 323.6088 (1986): 533-536.

[13] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th international conference on machine learning (ICML-10). 2010.

[14] Greene, William H. Econometric analysis. Pearson Education India, 2003.