Short Term Load Forecasting
Based on
Internet of Things (IoT)

SUBMISSION DATE: 25.03.2018

SUBMITTED BY:
Mahdi Uz Zaman (14101263)

Anisul Islam (13201044)

Nahid Sultana (14301139)

Supervisor:
Amitabha Chakrabarty, Ph.D

Assistant Professor

Department of Computer Science and Engineering

Declaration

We hereby declare that this thesis has been done based on the results we obtained from our work. Due acknowledgement has been made on text. This thesis neither in parts nor as a whole have been submitted previously by anyone of any institute or university for the award of any degree.

Signature of Supervisor

Amitabha Chakrabarty, Ph.D.
Assistant Professor
Department of Computer Science and Engineering
BRAC University

Signature of the Authors

Mahdi Uz Zaman
(14101263)

Anisul Islam
(13201044)

Nahid Sultana
(14301139)

ABSTRACT

In the era of internet every device is getting connected to the internet. Devices that are connected to the internet are called Internet of Things (IoT). In this paper we have assumed that IoT devices can share their power consumption history. Based on data points collected from real world environment we have conducted experiments to show that IoT can be used as a reliable backbone of a short term load forecasting system. In the experiment four machine learning algorithms Long Short-term Memory (LSTM), Support Vector Machines Regression(SVR), Decision Forest Regression with AdaBoost and Nearest Neighbors Regression were used to analyze the performance of the load forecasting system. In the experiment Long Short Term Memory Network has given comparatively better result than other three machine learning algorithm with a root mean square of 1.82.

Acknowledgement

First of all we would like to thank the Almighty for giving us mental strength and patience to complete the research.

Secondly, we would like to thank our supervisor Amitabha Chakrabarty. Without his help and guidance we could not have finished the project. In this past one year the path was not easy and filled with lots of ups and downs. Whenever we fall in problem and got depressed he was always there to show us the right direction. His motivation and inspiration helped us to achieve success. We are truly grateful to him.

We would like to thank our family members for their support and love throughout this tough journey. We would also like to thank our friends who have helped us with their valuable suggestions and experiences. Special thanks to those generous people who had spent their valuable time to make online tutorial which helped us a lot in this project.

Last but not the least, we would like to thank BRAC University for providing us good environment and lab support for conducting experiment for this research.

Contents

List of Figures

List of Tables

List of Abbreviations

ML- Machine learning

LSTM – Long Short-term Memory

CSV – Comma separated value

IDE – Integrated development environment

STLF – Short term load forecasting

MTLF – Medium term load forecasting

LTLF – Long term load forecasting

RMSE – Root mean squared error

CHAPTER 01

Introduction

Electrical energy generation and distribution is a complex and costly process. Efficient grid management plays a big role to reduce the cost of energy production. Grid management comprises of planning for load demand, maintenance of generation units, supply lines and efficient load distribution across the supply line. Therefore an accurate load forecast will increase the efficiency of planning process of a power generation company. Power generation companies do their plan based on data collected manually. Therefore real time prediction is not possible. If data can be collected in real time, forecasting in real time will be possible. Strong and reliable Internet infrastructure are already present. Every device we use in our daily life are gradually getting connected to the internet to facilitate smart home technologies like Google Home [1], Amazon Alexa etc. A device connected to internet usually treated as IoT. In general a device with sensors, microprocessor or microcontroller which can connect to the internet, send and receive information through internet is called Internet of Thing (IoT) [2]. If the devices are configured to send energy uses data to the internet, these data can be used to give real time forecasting. In this paper we have shown that real time load forecasting is possible with the help of IoT and state of the art machine learning algorithms LSTM Network, Nearest Neighbors Regression, Support Vector Regression and Decision tree Regression with AdaBoost.

1.1 Motivation

First of all, increased demand of electricity is creating pressure on production companies as well as natural resources. We know natural resources which are used to produce electricity are limited in nature. Secondly, the byproduct of electricity generation is pollution. Again, cost for producers and consumers are increasing day by day. Therefore to ensure sustainable development research communities have shown great interest on how to reduce electricity demand by efficient use of electricity. One of the important of methods that is used to facilitate efficient use of electricity is load forecasting [3]. With the help of load forecasting, producers can tune their production plan and consumer can optimized their electricity consumption. Existing forecasting system relies on data collected from production and

distribution unit. We have shown that with the help of IoT load forecasting can be done in more easy, convenient and reliable way.

1.2 Contribution Summary

The main contribution of this project is to show that IoT can be used as a reliable backbone of a load forecasting system. To support our claim we have tested our system with real world datasets. Based on this dataset we have done empirical comparison and performance evaluation of four machine learning algorithm. This system will help home user to reduce their power consumption by early warning of future power use. This will also help the power generation company to meet their demand efficiently by planning ahead of time.

1.3 Thesis Outline

In this paper, Chapter 2 provides the literature review in details including the algorithms and techniques used in the system. Proposed model including the algorithms and techniques are discussed in Chapter 3. Results and analysis are presented in Chapter 4. Lastly Chapter 5 gives the conclusion and future work.

CHAPTER 02

Literature Review

Load forecasting plays an important role to the efficient use of electricity as well as efficient production and distribution. Power load forecasting is categorized in three categories [4]. They are short term load forecasting (STLF), medium term load forecasting (MTLF) and long term load forecasting (LTLF). These categorization is depend on the range of future time taken in to consideration to be predicted. Prediction process which give prediction day or week ahead is called short term load forecasting (STLF). Medium term load forecasting (MTLF) system are built for month ahead prediction and when years ahead predictions are needed long term load forecasting (LTLF) system are incorporated. In this project we have built and tested a system which can predict a day ahead forecasting. In this chapter we will discuss about the algorithms used to build the short term load forecasting system. In the past researchers have proposed different types of methods for load forecasting. We will also discuss about some of them in this chapter.

Kong, W. contributed in deep learning based method [5] with appliance behavior learning for meter level load forecasting which demonstrated an advantageous performance through extensive comparison with other predictors. According to this paper, if we can learn the lifestyle pattern of certain resident can help us achieve better metering forecasting. His work showed that the using appliance measurements in training data can improve the forecasting accuracy. Contextual variables like temperature, humidity, day of the week and special events are taken into consideration in this method for better forecasting performance. In this paper individual load forecasting is done using LSTM. Long short-term memory (LSTM) is one of the RNN structure, the specialty of this RNN is sequence learning. It maintains a memory cell in its structure to remember important state in past to reset the memory cell it has a forget gate. As mentioned above the learning lifestyle pattern can be done if appliance level consumption are directly measured which assists in interpreting in the forecasting. So instead of serving aggregated data to the LSTM the inputs are all available major appliance energy sequence to train the predictor. Here the dataset is taken from a Canadian household and its 19 appliance for a year. Then the current reading is converted into Ampere hour for every 30 minutes to imitate the smart meter data. The appliance chosen for appliance learning are clothes dyer, clothes washer, dishwasher, heat pump, television and wall oven which are manually operated. For resident behavior learning this approach used the

measurement of both the whole house-hold consumption and the selected appliance from the past several time interval until the current time as inputs. The consumption forecast of subsequent time interval is the output. To compare the performance of this proposed method feed forward neural network (FFNN) and K-nearest neighbor (KNN) is used. The lowest benchmark is set by empirical mean which is the forecasting value of the statistical mean given by the time of the day and day type. Here "look-back" input scheme, a system level load forecasting which uses measurement of same time interval of the past few days also compared referred with suffix "D" suffix "WA" is used to label test cases that use extra appliance measurement in training data and suffix "W" for whole house measurement only. This paper concluded showing that LSTM-WA outperformed all other methods. And LSTM-WA with two look back interval achieved the best overall MAPE scores and the second best LSTM-W predictor with a 4.24% MAPE margin. KNN and FFNN the version of extra appliance data gave better result compared to whole house consumption. In conclusion, the LSTM based forecasting framework gives better accuracy when consumption sequence of major appliance is available.

Ghulam and Angelos, worked on the applicability and compared the performance of Feed-forward deep neural network (FF-DNN) and recurrent deep neural network (R-DNN) models on the basis of accuracy and computational performance in the context of time wise short term forecast of electricity [6]. Analyzing the data on the time and frequency domain independently and subsequently frequency domain components are transformed back to the time domain. The parameter which are taken into consideration are weather, time, holidays, working days, and lagged load and data distribution effects. This paper collected the dataset from ISO New England for duration 2007-2012. The load consumption is recorded at the end of every hour of a day and the whole dataset consisted of 52600 records that represented data of 6 states of New England, USA. The changing behavior of the dataset is captured during analysis in time domain and effects that were captured are temperature effect, working and non-working day's effect, time effect, lagged load effect and data distribution effect. After time domain extraction the data is further analyzed in frequency domain. The random signals of time domain are converted to different frequencies that are stable and easily predictable which improves accuracy. Fast Fourier is performed to determine the dominant frequencies and the one with higher magnitude represent the dominant frequencies. Here evaluating the proposed models 43824 records from 2007-2011 are used in training dataset and 24 and 168 records for days and weeks for test dataset. The RMSE, MAE and MAPE are calculated for four seasons

of 2012 and the 5 features extracted from the original features are taken into consideration for better accuracy. The result of considering only time domain in both FF-DNN and R-DNN varied due to temperature variation in different seasons. The MAPE is 1.30% for R-DNN and 1.42% for FF-DNN in a year. On the other hand the error are much lower and the accuracy is improved in frequency domain analysis where MAPE is 0.067% and 0.057% for FF-DNN and R-DNN respectively. So based on the analysis it was shown that weather, time, holidays, lagged load and data distribution have most dominant factors and the TF features can be utilized for load forecasting.

Papia Ray, Santanu Sen and A.K. Barisal presented two hybrid methodologies based on discrete wavelet transform (DWT) in combination with ANN or SVR for Short Term Load Forecasting (STLF) using feature selection [7]. This method was done with the data taken from a particular area of New Delhi for a particular month. The data is taken from December 1 to February 28. Temperature, humidity, dew point and load consumed for a particular day at a particular hour are also taken into consideration. Here data from December 5 to January 31 are taken as training data, from February 1 to 28 are used as validation set and the 4days data are taken as test set data. Here the feature selection is done through Forward Feature Selection (FFS). The analysis was done in two ways one using FFS and other without using FFS and it showed that the one done with FSS gave a better result. These analysis was done on monthly based, weekly based and daily based and among two hybrid methods DWT-SVM showed an error of 0.1% and DWT-ANN showed error of 0.6% which concluded that DWT-SVM showed better result than DWT-ANN.

Taking in consideration of the above mentioned work we have implemented an IoT based load forecasting system. The core algorithm of the forecasting system is a machine learning algorithm. To select best performing algorithm we have tested performance of several machine learning algorithm with a new dataset called “The UK-DALE dataset” [8].

2.1 Load Forecasting with Machine Learning

Machine learning is the ability of a machine to do certain task without being explicitly programed [9]. One of the branches of machine learning is supervised learning. Supervised learning is process of turning experience into expertise [10]. In another way when machine learn from previous incidents and take decision on unknown incidents is called supervised learning. Load forecasting is a complex process due to its nonlinear and always changing

nature. If we can teach a machine the user pattern of a certain device which consume electric power a good machine learning algorithm can tell the probability of future occurrence. Machine learning algorithm produce hypothesis after seeing the data. As hypothesizes are constructed based on datasets machine learning hypothesis are adaptable with different scenario and circumstances. Therefore we have choose machine learning approach for load forecasting.

2.2 Time Series Analysis with Machine Learning

In time series data points are collected sequentially with respect to time [11]. Three are mainly two type of time series. One is discrete time series, where data point are collected with a fixed interval of time. One the other hand in continuous time series data points are taken continuously without any interval. In the definition of discrete time series it is said that difference between two data points must be maintained throughout the whole datasets. The datasets we have used in this project is discrete time series. Time series helps to analyze the user pattern of electrical devices with respect to time. Time series analysis is a process where time series datasets are analyzed to extract features. In this paper we have used machine learning approach to analyze time series datasets. We cannot use a time series dataset directly for a machine learning algorithm. Different algorithm requires special kind of data preparation. In equation 1, y is predicted power consumption in a given time (t) and x is the features of time (t-n). Based on features x a machine learning algorithm will predict y. In this paper we have used n=1. That means to predict the power consumption of day (t), we have used features of day (t-1).

$$y_t = x_{t-n} \tag{1}$$

2.3 Long Short-term Memory Network (LSTM)

LSTM Network is built based on the basic concept of Recurrent Neural Network (RNN). Before going to the details description of LSTM a short summary on RNN is given below.

2.3.1 Recurrent Neural Network

A Recurrent Neural Network [11] is a combination of numbers of identical feed forward neural network. This combination is made by copying the same network sequentially with

respect to time steps. Each copy of the network can pass information to the newer copy of the network. Figure 1 demonstrate how a RNN go through time steps by coping itself has shown.

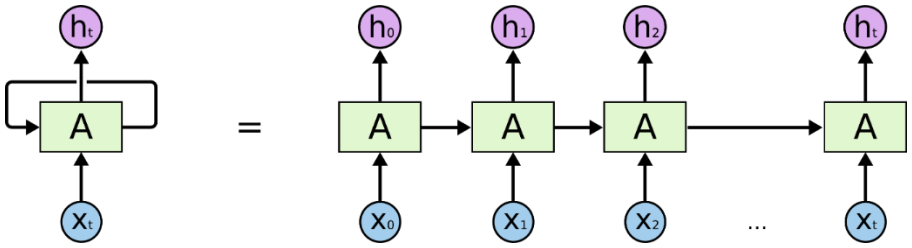

Figure 1. Recurrent Neural Network [12]

2.3.2 RNN Cell

Unlike neuron of a neural network, RNN has cell. A RNN cell itself contain a neural network. In Figure 2 a RNN cell has been shown. In Figure 1 box denoted by "A" is a RNN cell. In each time step RNN cell has a cell state.

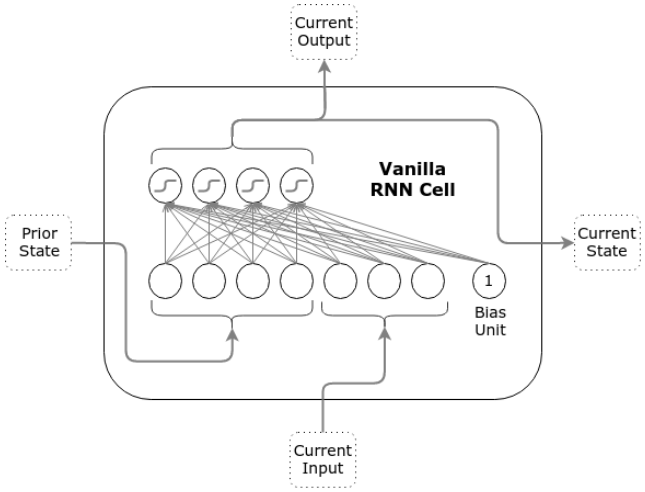

Figure 2. RNN Cell [13]

$$h_t = \sigma(W.[h_{t-1}, x_t] + b) \quad (2)$$

Output of RNN cell is computed by the equation 1 [14]. Here h_t is the output of the current state. Previous state h_{t-1} and current input x_t is multiplied by a weight W and added to bias b. Then the result is passed through a sigmoid activation function. Vector dimension of h_{t-1} and x_t is same.

2.3.3 Vanishing Gradient of RNN

In Figure 2 a sequence of RNN Network has been shown. If the sequence is too long then the network faces vanishing gradient problem due to backpropagation through time. Due to this problem RNN Network cannot remember an occurrence which happened long time ago. This is the major drawback of RNN Network [15]. This is also called long term dependency problem.

2.3.4 LSTM Network

Sepp Hochreiter and Jürgen Schmidhuber published a paper [16] in 1997 called "Long Short-term Memory" to address the drawbacks of basic RNN Network. They proposed a new architecture of RNN Cell. They named the new architecture as LSTM.

2.3.5 LSTM Cell

Specialty of a LSTM cell is its memory and three gates. LSTM uses the memory to remember from very deep down the sequence. It uses the gates to control the flow of information from memory and to the memory. In Figure 3 a schematic of LSTM cell is given. Input of LSTM Cell:

- Previous cell State (Ct_{-1}): Memory of previous state is forwarded to the current state (Ct). Which will then added to the current state by increment. Process of increment is given in the description of forget gate.
- Previous cell output (h_{t-1}): Cell output of previous time step is taken as input in current time step.
- External input (x_t): Neural network output of current time step. This output is optional. Depend of the need it sometimes give output an sometimes not.

Output of LSTM Cell:

- Current cell state (Ct): Current cell state is forwarded to future time step through cell state output.
- Cell output (h_t): After squashing cell state through tanh activation function, cell output is forwarded to future time step.
- External output (h_t): Optional Neural Net output.

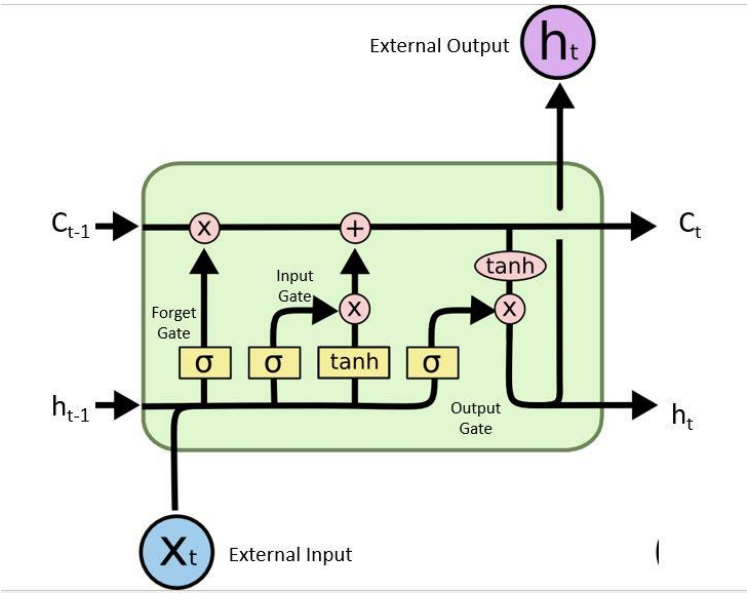

Figure 3. LSTM Cell [12]

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \quad (3)$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \quad (4)$$

$$\check{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \quad (5)$$

$$C_t = i_t * \check{C}_t \quad (6)$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \quad (7)$$

$$h_t = o_t * \tanh(C_t) \quad (8)$$

- W_f : Weight for forget gate layer
- b_f : Bias for forget gate layer
- i_t : Output of input gate layer
- W_i : Weight for input gate layer
- b_i : Bias for input gate layer
- $\check{C}_t$: Candidate value
- W_c : Weight of tanh layer
- b_c : Bias of tanh layer
- W_o : Weight of output gate layer

- b_o : Bias of output gate layer

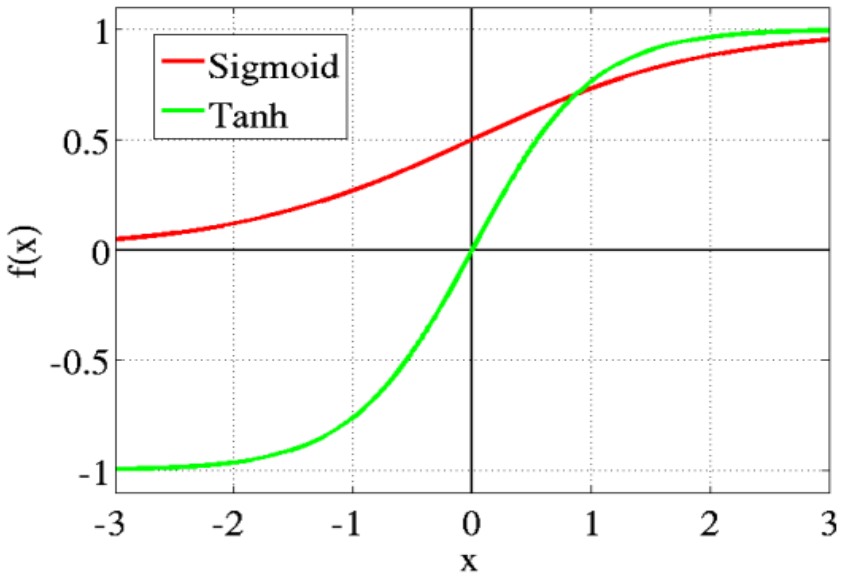

Figure 4. Sigmoid and Tanh function graph

Gates of LSTM Cell:

- **Forget Gate (f_t):** Overwhelming unnecessary information causes information morphing to the cell state, as known as memory. Forget gate shorts out most relevant information by sigmoid layer. In Figure 4 a sigmoid function is given. It squashes the output between 0 and 1. 1 means completely remember and 0 means completely forget. Equation 3 [16] shows that with respect to previous cell output and current input it calculate which part of the previous information need to remember in the current state.
- **Input Gate (i_t):** It is also known as write to the memory. When LSTM cell gets new external input it needs to decide which part of the memory it will overwrite with new value. First with the help of sigmoid layer it decides which values it needs to update. Weighted external input, previous cell output and bias is passed to sigmoid layer according to Equation 4 [16]. Then a tanh layer select a vector of candidate values by Equation 5 [16] for the selected values by Equation 4. In Figure 4 a tanh function is given. A tanh function squashes input between -1 and 1. After multiplying output of Equation 4 and Equation 5 element wise we get a new cell state or memory state (Ct), Equation 6.
- **Output Gate (o_t):** Unlike RNN cell output, LSTM cell does not output the exact copy of cell state. Instead it outputs a filtered version of the cell state. A sigmoid gate Equation 6 decide which part of the memory will be forwarded to the next state. Present

cell is squashed by a tanh layer to scale it between -1 and 1 multiplied elementwise with output of equation 6 [16] to produce cell output, Equation 7 [16].

The ability of selectively read, write and remember the events happened in numbers of previous time steps makes LSTM a robust algorithm for sequence learning.

2.4 K-Nearest Neighbors Regression

K-NN algorithm saves all convenient cases to predict the targeted value based on the similar values. K-NN is not a new technique, it has been used in 1970's for statistical estimation and pattern recognition as non-parametric technique.

2.4.1 Algorithm

KNN regression is used to calculate average of targeted value of k nearest neighbors. Using inverse distance weighted average of the k nearest neighbors can also be calculated. KNN classification and KNN regression uses the same distance functions. With the help of the following functions distance between neighbors are measured [17]:

Euclidean

$$\sqrt{\sum_{i=1}^{k}(x_{i-}y_i)^2} \tag{9}$$

Manhattan

$$\sum_{i=1}^{k}|x_i - y_i| \tag{10}$$

Minkowski

$$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q} \tag{11}$$

Hamming Distance

$$D_H = \sum_{i=1}^{k}|x_i - y_i| \tag{12}$$

$$if\ x = y\ then\ D = 0$$

$$if\ x \neq y\ then\ D = 1$$

These equations can only be used for continuous variables. For categorical variables Hamming distance must be used. This measures the number of instances where different corresponding symbols are in two strings of equal length.

Inspecting the data the ideal value for K is chosen. With a large K value the noise is reduced but it becomes harder to detect the distinct features. To determine good K value using independent data set to validate K values, cross- validation is an ideal way. The ideal K for most datasets is 10 or more which produces better results than 1-NN.

2.4.2 Standardized Distance

Standardizing the training set can overcome the difficulty to calculate distance measures directly from the training set where there is a mixture of numerical and categorical variables.

$$X_s = \frac{X - Min}{Max - Min} \tag{13}$$

Using the standardized distance on the same training set, the unknown case returned a different neighbors which is not a good sign of robustness.

2.5 Support Vector Machine – Regression

Support Vector Machine can also be used as a method of regression, keeping all the fundamental elements intact that designate the algorithm (maximal margin). With scarcely trivial distinction, the Support Vector Regression utilizes the same postulates as the SVM for categorization. Firstly, because it has indefinite possibilities, the prediction of the resulted real number becomes perplexing. In case of regression, a margin of tolerance (epsilon) is allocated in approximation to the SVM which would have already arise from the problem. Aside from this, there is another complication. Thereby the algorithm is further more problematic to be accepted. Nevertheless, the gist stays the unchanged: diminishing error, singularizing the hyperplane which maximizes the margin, realizing the part of the error is tolerated. The followings are mathematical formulation of the kernels used in SVR [18]:

Linear SVR:

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*).(x_i, x) + b \tag{14}$$

Non-Linear SVR

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*).\big(\phi(x_i), \phi(x)\big) + b \tag{15}$$

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*).K(x_i, x) + b \quad (16)$$

Polynomial

$$K(x_i, x_j) = K(x_i, x_j)^d \quad (17)$$

Gaussian Radial Basis function

$$K(x_i, x_j) = \exp\left(-\frac{\left||x_i - x_j|\right|^2}{2\sigma^2}\right) \quad (18)$$

2.6 Decision Tree – Regression with AdaBoost

Decision tree establishes regression models in the form of a tree structure. It simultaneously jots down dataset onto smaller and smaller subsets and aligned decision is incrementally established. The end result is a tree with decision nodes and leaf nodes. A decision node has two or more sectors, each indicating values for the feature tested. A numerical target is illustrated by leaf node. The highest decision node in a tree which becomes equivalent to the leading predictor is called root node. Both categorical and numerical data can be conducted by decision trees [19]. The key advantages of trees are that they can be rapidly instructed (say, as distinguished between neural networks) and are non-parametric. The major disadvantages are that the space has limits that are parallel to the characteristic axes and representation based on powers or products of the features are disapproved. Making decision surfaces that are oblique to the axes are possible and using so called oblique decision trees [Bradley and Utgoff (1995), Ittner and Schlosser (1996), Murthy et.al. (1993). Mouth et.al. (1994)] [20] and as a matter of fact CART has that option. Moreover, in lieu of the input to the tree being the features, we could have maintained both the products of features and features raised to some powers. All these alternatives make the constructing of trees prolonged.

CHAPTER 03

Proposed Model

The IoT based Short Term Load forecasting system is consist of IoT devices and a central processing unit. IoT devices are configured to upload power usage data to the server. The central processing unit is configured to do calculation based on a selected machine learning algorithm. Based on the calculation and learning process the unit will give prediction. In this chapter we will give details description about the proposed model and process of implementing the model.

3.1 Process

There are four steps to give Short Term Load Forecasting. In Figure 5 a pectoral view of proposed model is given.

- Data collection from IoT devices
- Preprocessing and filtering
- Training machine learning model
- Load Forecasting

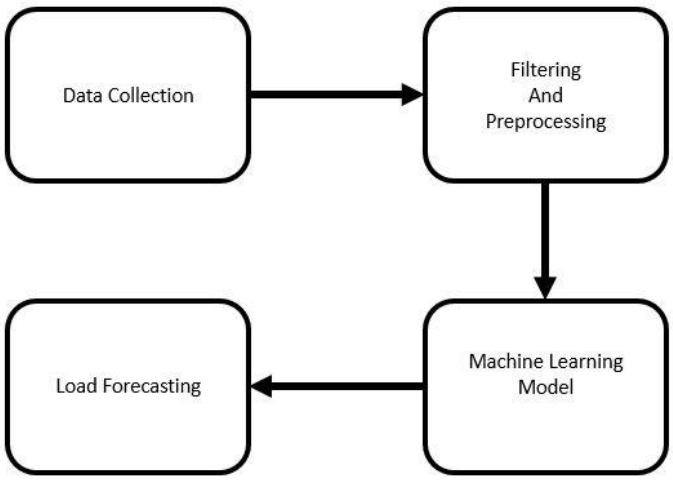

Figure 5. Proposed Model

3.2 Data Collection

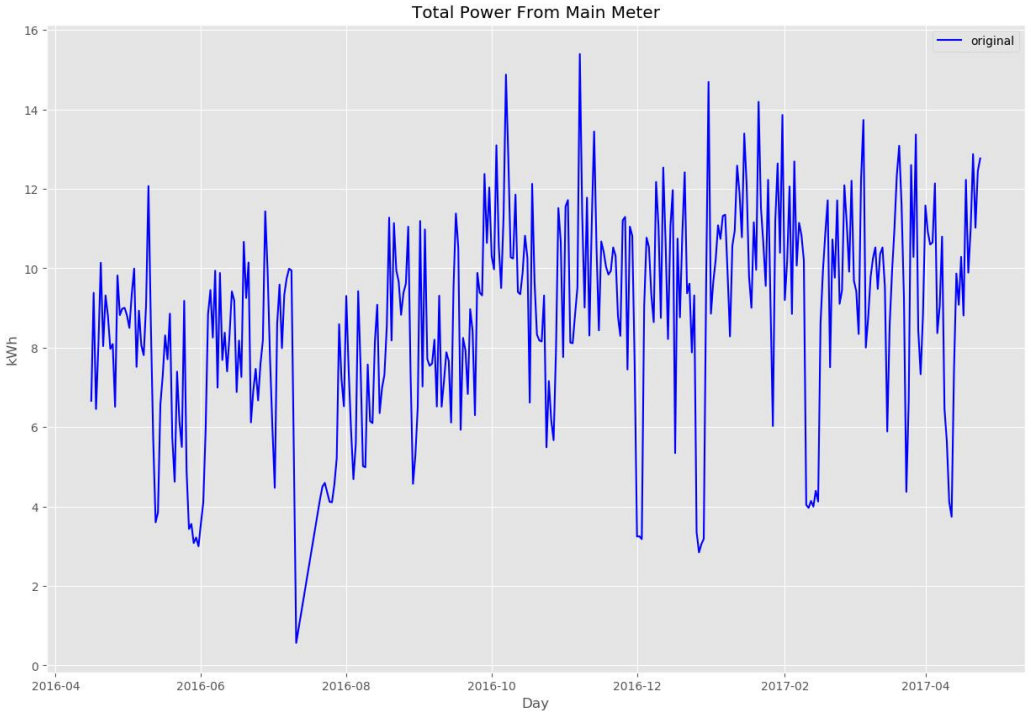

Figure 6. Total Power Consumption of 1 year

Main idea of this project is forecast total power consumption based on the data collected from the IoT devices. Due to limitations of time and resources we could not configure devices to upload power uses data to the internet. In this project we have used "The UK-DALE dataset" created by Jack Kelly & William Knottenbelt [8]. This datasets contains appliance level disaggregated power consumption record as well as aggregated whole house power consumption record. In Figure 6 whole house power consumption of last one year is given. We assumed that all the appliances in this datasets are IoT devices. In Table 1 a short overview of the whole dataset is given. For training and testing we have used data of house_1 because it contains maximum number of appliances. Also they have given more emphasis on recording house_1 data. In Figure 7 shows that house_1 data are more consistent than other houses.

Table 1. Summary of The UK-DALE dataset

House	1	2	3	4	5
Number of occupants	4	2	2	2	2

Description of occupants	2 adults and 1 dog started living in the house in 2006. One child born in 2011. Second child born in 2014.	2 adults. 1 at work all day; the other sometimes home	1 adult and 1 pensioner		2 adults
Total number of meters	54	20	5	6	26
Date of first measurement	2012-11-09	2013-02-17	2013-02-27	2013-03-09	2014-06-29
Date of last measurement	2017-04-26	2013-10-10	2013-04-08	2013-10-01	2014-11-13

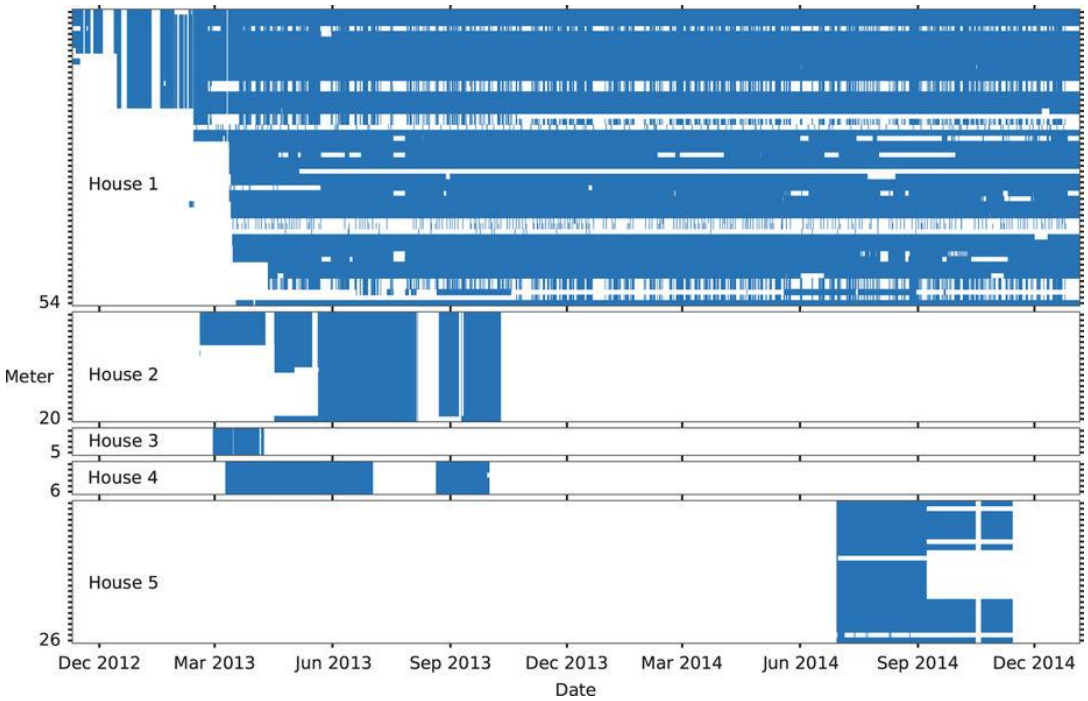

Figure 7. Summary of The UK-DALE dataset

3.3 Preprocessing and Filtering Data

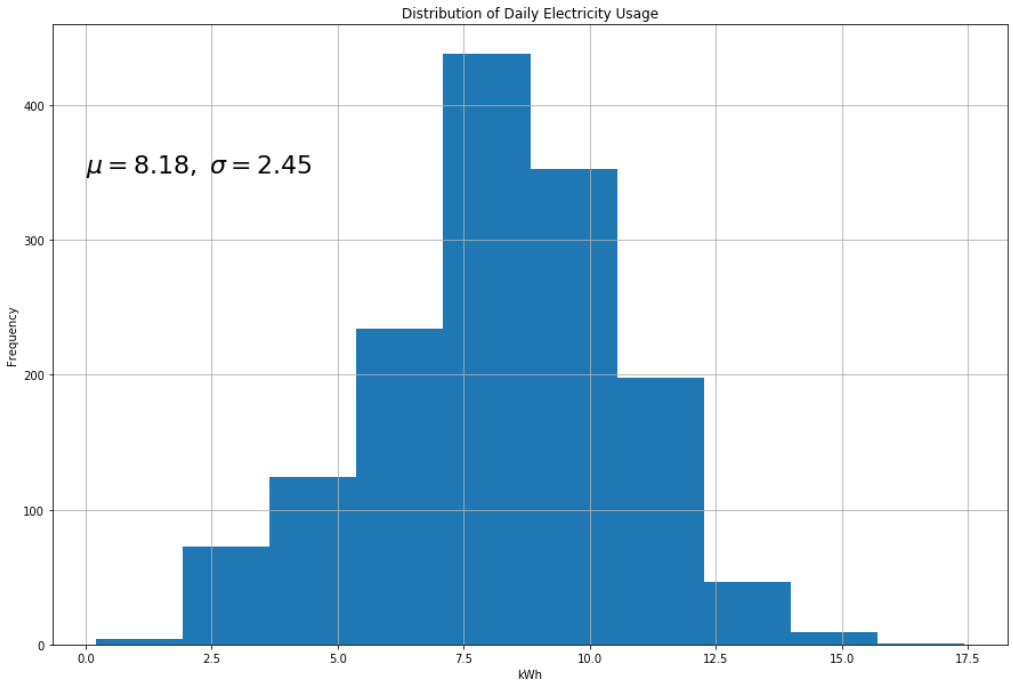

Figure 8. Distribution Graph of Total Power Consumption

Noise and misleading data in any dataset are bad for any model to train on. A misleading dataset will eventually produce a hypothesis which will not do well in unseen data. Therefore noise cancellation has done with great care. The dataset contains UML configuration file for every houses. The file has details description of meter devices and appliances. Each appliance has upper bound and lower bound of power consumption. Any power consumption beyond that limit is considered as noise or bad reading. In Table 2 upper and lower bound of every device recorded in house_1 is given. The main dataset contains five folder. Each folder correspond to each house. Under each house numbers of CSV files according to number of devices are given. Each CSV file contains records of power consumption with time. In each CSV file time is give in format of UNIX time epoch. Interval data recording is six seconds. In Figure 8 a screenshot of first 10 rows of channel_6 which is the records of power consumption of dishwasher is given. In Figure 8 we can see CSV file contains two column and column_0 contains time and column_1 contains power consumption record.

	1
0	
1352500098	1
1352500104	1
1352500110	1
1352500116	1
1352500122	1
1352500128	1
1352500134	1
1352500140	1
1352500146	1
1352500152	1

	1
2012-11-09 22:28:18	1
2012-11-09 22:28:24	1
2012-11-09 22:28:30	1
2012-11-09 22:28:36	1
2012-11-09 22:28:42	1
2012-11-09 22:28:48	1
2012-11-09 22:28:54	1
2012-11-09 22:29:00	1
2012-11-09 22:29:06	1
2012-11-09 22:29:12	1

	Ws
2012-11-09 23:41:19	161
2012-11-09 23:55:26	150
2012-11-09 23:58:06	2380
2012-11-09 23:58:12	2332
2012-11-09 23:58:18	2313
2012-11-09 23:58:24	2350
2012-11-09 23:58:30	2332
2012-11-09 23:58:36	2367
2012-11-09 23:58:42	2338
2012-11-09 23:58:48	2326

Figure 9. Data Preprocessing

We have used pandas, a powerful library written in python for data analysis and manipulation. Steps for filtering every CSV file is given below.

- With the help of pandas we converted UNIX time epoch to human readable date and time.
- We dropped all the data record that do not comply with the bound given in Table 1. In Figure 10 a screen shot of noise free reading is given. Now we can say that the dataset does not contain any noise according to Table 2. Power consumption records are taken via external meter Figure 11. The meters require some power to operate and not factory standard. Therefore error is obvious in their reading. In IoT devices power consumption reading capability will be integrated in their internal circuit and will not face this kind of error.
- Goal of this project is to give a day ahead forecast. Therefore we do not need six second interval datasets. Six second datasets are then resampled by day. Then total power consumption of the day is converted into kWh, Figure 10.

- House power consumption has good relation with weather condition. Therefore we have added average temperature, average humidity and average wind speed of every day.

- In Figure 9 distribution graph of total power consumption is given. From this distribution we have dropped lower than (mean - 2 *σ) and greater than (mean + 2 *σ). According to normal distribution probability of any event outside 2 σ is less than 5%.
- After completing aforementioned steps all the CSV files are concatenated into a single CSV file. Then the date and time was again converted into UNIX time epoch. Reason of converting is, date time format changes in different computer due to different version of software. UNIX time epoch is an integer number. As a result date and time remain intact in every computer that want to work on the dataset. In Figure 12 a partial screen shot of final CSV file is given.

	Ws	KWh
2017-04-16	30682980	8.52
2017-04-17	29304360	8.14
2017-04-18	17066376	4.74
2017-04-19	42035688	11.68
2017-04-20	17977860	4.99
2017-04-21	13381992	3.72
2017-04-22	60012144	16.67
2017-04-23	58531536	16.26
2017-04-24	30950028	8.60
2017-04-25	30414168	8.45

Figure 10. Dataset after converting to kWh per Day

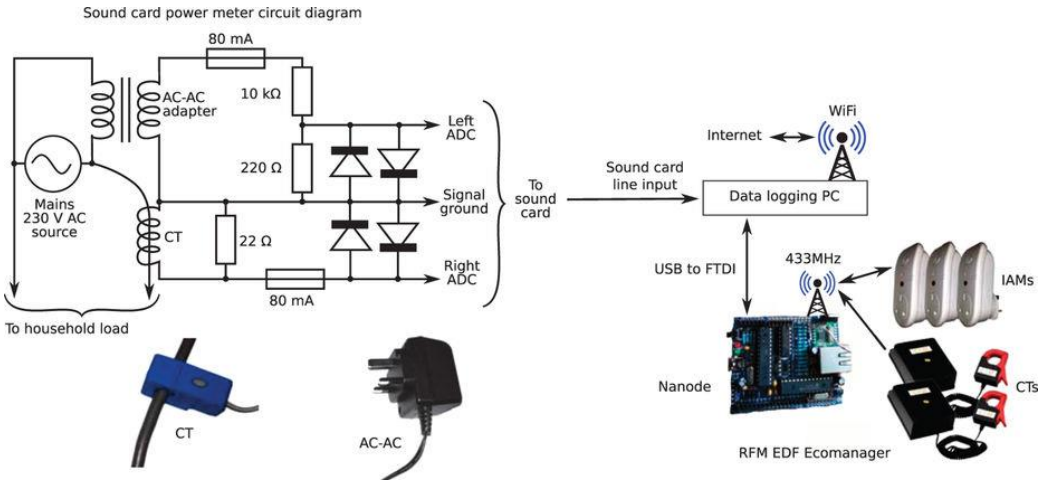

Figure 11. Smart plug and Data logger

Table 2. List of Devices

Channel ID	Name	Min Power (Watt)	Max Power (Watt)	Type
2	boiler	70	4000	Apparent
3	solar_thermal_pump	43	4000	Apparent
4	laptop	70	4000	Active
5	washing_machine	20	4000	Active
6	dishwasher	10	4000	Active
7	tv	10	4000	Active
8	kitchen_lights	50	4000	Apparent Sub meter of channel 25
9	htpc	20	4000	Active
10	kettle	2000	4000	Active
11	toaster	1000	4000	Active
12	fridge	50	4000	Active
13	microwave	200	4000	Active
14	lcd_office	40	4000	Active
15	hifi_office	9	4000	Active
16	breadmaker	500	4000	Active
17	amp_livingroom	25	4000	Active
18	adsl_router	6	4000	Active
19	livingroom_s_lamp	16	4000	Active
20	soldering_iron	50	4000	Active
21	gigE_&_USBhub	5	4000	Active
22	hoover	1200	4000	Active
23	kitchen_dt_lamp	13	4000	Active
24	bedroom_ds_lamp	26	4000	Active
25	lighting_circuit	40	4000	Apparent
26	livingroom_s_lamp2	86	4000	Active
27	iPad_charger	7	4000	Active
28	subwoofer_livingroom	15	4000	Active
29	livingroom_lamp_tv	13	4000	Active

30	DAB_radio_livingroom	300	4000	Active
31	kitchen_lamp2	20	4000	Active
32	kitchen_phone&stereo	5	4000	Active
33	utilityrm_lamp	35	4000	Active
34	samsung_charger	4	4000	Active
35	bedroom_d_lamp	45	4000	Active
36	coffee_machine	1000	4000	Active
37	kitchen_radio	2	4000	Active
38	bedroom_chargers	2	4000	Active
39	hair_dryer	1600	4000	Active
40	straighteners	170	4000	Active
41	iron	1700	4000	Active
42	gas_oven	11	4000	Active
43	data_logger_pc	12	4000	Active
44	childs_table_lamp	14	4000	Active
45	childs_ds_lamp	10	4000	Active
46	baby_monitor_tx	15	4000	Active
47	battery_charger	20	4000	Active
48	office_lamp1	14	4000	Active
49	office_lamp2	10	4000	Active
50	office_lamp3	7	4000	Active
51	office_pc	100	4000	Active
52	office_fan	20	4000	Active
53	LED_printer	400	4000	Active

3.4 Training Machine Learning Model

3.4.1 Long Short-term Memory Network

LSTM networks are renowned for their ability to remember pattern and sequence. Human behavior tends to be repetitive. From this intuition we used LSTM Network to learn the behavior pattern of power usages. Also, unlike normal datasets time series has a complexity of order dependence between items and sequence. Long Short-term Memory Network were

made to deal with these kind of sequence dependencies. To build LSTM Network, programming was done in Python 3.5. Python framework especially Pandas was used to read dataset from a CSV file. To make the dataset suitable for LSTM, numPy was used to reshape the dataset. We used TensorFlow with a rapper called Keras.

	A	B	C	D	E	F	G	H	I
1		boiler	solar_ther	laptop	washing_r	dishwashe	tv	htpc	kettle
2	1363478400	0.467	0.128	0	2.715	0	0.273	0.208	0.4
3	1363564800	0.611	0.106	0	0	0	0.096	0.129	0.4
4	1363651200	0.555	0.13	0.01	0	0.921	0	0.021	0.3
5	1363737600	0.536	0.053	0	0	0	0.246	0.256	0.3
6	1363824000	0.506	0.094	0	1.117	0.999	0.246	0.155	0.
7	1363910400	0.729	0.124	0	0.546	0	0.107	0.246	0.4
8	1363996800	0.743	0	0	0	0	0.337	0.269	0.2
9	1364083200	0.864	0	0	0.687	0.426	0.173	0.125	0.2
10	1364169600	0.857	0.009	0.046	0	0.577	0.084	0.246	0.
11	1364256000	0.695	0.13	0.014	0.576	0.737	0.387	0.344	0.
12	1364342400	0.682	0.173	0	0.564	0.21	0.186	0.342	0.2
13	1364428800	0.634	0.129	0	0.704	0	0.218	0.261	0.
14	1364515200	0.144	0.216	0	0	0	0	0.08	0.1
15	1364601600	0	0.107	0	0	0	0	0.065	
16	1364688000	0	0.067	0	0	0	0	0.003	
17	1357257600	0.212	0.177	0	0	0	0.092	0.125	
18	1359936000	0.523	0.302	0.058	0.605	1.061	0.478	0.333	0.4
19	1362355200	0.681	0.066	0	1.872	0.978	0.399	0.286	0.5
20	1365033600	0.63	0.054	0	0	0	0.37	0.315	0.3

Figure 12. Partial screen shot of Final Dataset

Input to a LSTM Network is 3D matrix. Datasets we have used in this project is 2D matrix which consist of columns and rows. Number of columns corresponds to dimension of feature vector and number of rows corresponds to number of data points. Input of LSTM Network consist of another dimension of the matrix, which corresponds to time steps. LSTM network use this time steps to keep track of the previous occurrence. In Figure 13 graphical view of the input of a LSTM Network is given. Structure of LSTM Model is described below Figure 14.

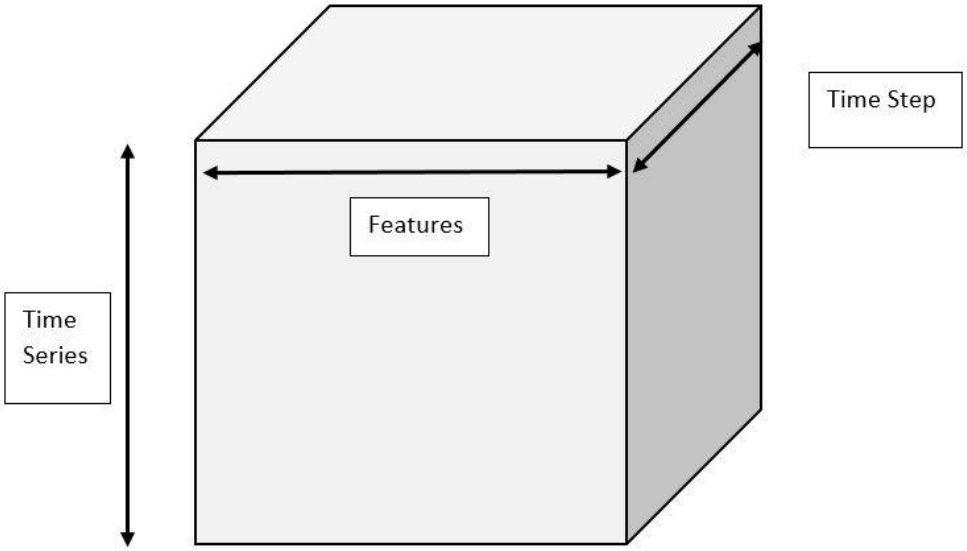

Figure 13. Input Shape of LSTM

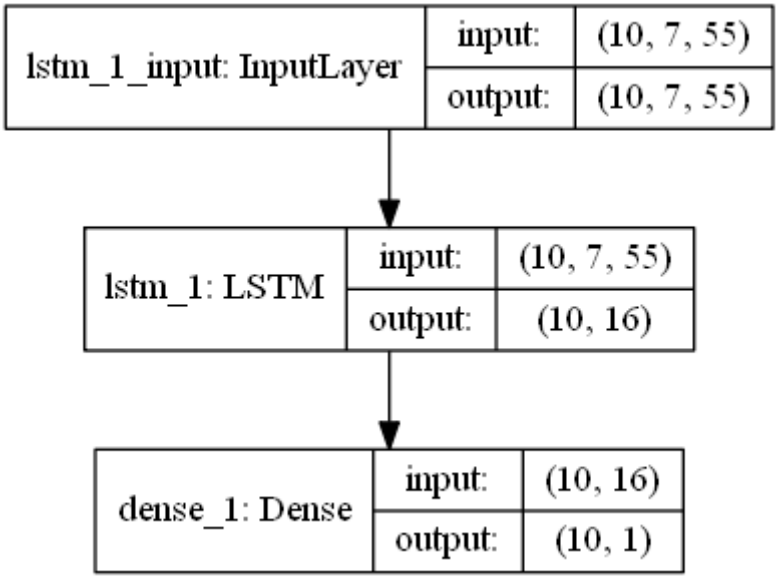

Figure 14. Structure of Neural Network

In input layer is used to take output from outside and pass it to LSTM cell. In this layer input and out shape are same.

- Input and output Shape: (Batch Size * Time Step * Input Vector Dimension)
    - Batch Size: 10 sequence of samples

    - Time Step: 7. Every sequence consist of today and previous 6 days of power consumption reading
    - Input Vector Dimension: 55

LSTM Cell take input of a 55 dimensional vector and give output of a 16 dimensional vector.

- Number of LSTM Cell: One.
- Batch Input Shape: (Batch Size * Time Step * Input Vector Dimension)
    - Batch Size: 10 sequence of samples
    - Time Step: 7. Every sequence consist of today and previous 6 days of power consumption reading
    - Input Vector Dimension: 55
- Batch Output Shape: (Batch Size * Time Step * Input Vector Dimension)
    - Batch Size: 10 sequence of samples
    - Time Step: 10. Every sequence consist of today and previous 10 days of power consumption reading
    - Output Vector Dimension: 16
- Activation: tanh
- Recurrent activation: Hard sigmoid
- Return Sequence: True

Output Layer of this network consist of one dense layer which take 16 dimensional vector as an input and give output of one dimensional vector.

To prevent over fitting of the model a threshold was given. A program always keep track on difference between loss of training set and loss of test set. When loss of training set is decreasing and loss of test set is increasing and difference is bigger than threshold, it stops training the model.

3.4.2 K-Nearest Neighbors Regression

Neighbors-based regression uses continuous data labels rather than discrete variables. The query point is appointed a label which is the mean of the labels of nearest neighbors.

Two different neighbors regressor are implemented by scikit-learn those are $KNeighborsRegressor$ where the implementation is done learning the k nearest neighbors of each query and k is user specified integer. And $RadiusNeighborsRegressor$ is implemented

learning about the neighbors in a fixed radius r of the query point and r is user floating-point value.

Uniform weights are used by the basic nearest neighbors that is the classification of query point is contributed uniformly by each point in local neighborhood. In some conditions, to weight points in a way that the nearby points accord more to regression than faraway points can give us advantage. This can be done by using the keyword $weights$. $weights = 'uniform'$ is the default value assigned to all points with equal weights and from the query points the weights proportional to the inverse of distance is allocated by $weight = 'distance'$. In other way, to compute the weights user can define a function of the distance.

Brute Force: In machine learning the committed area of research is fast computation of nearest neighbors. The most raw neighbor search implementation is the brute-force computation of distances among all pairs of points in the dataset. $O[DN^2]$ is scaled where N is the number of samples and D is dimensions. For small data samples brute-force neighbor search can be very moderate. However with the increase in the number of samples N the brute-force approach becomes unworkable. The search by brute-force neighbors in the class $sklearn.neighbors$ is limited to the keyword $algorithm = 'brute'$ and computation is done using the routines available in $sklearn.metrics.pairwise$.

K-D Tree: Tree based data structure have been invented to meet the computational ineffectual of brute-force approach. By encoding aggregate distance information from the sample these data structure reduces the number of distance calculation. For example if point A is very far from point B and point B is very near to point C then the points A and C are very far, this can be found without separately calculating their distance. This way the cost of nearest neighbors can be minimized to $O[DN\log(N)]$. This is a notable upgrade over brute-force for large N [21].

KD tree data structure derives two-dimensional Quad-trees and 3-dimensional Oct-trees to an arbitrary number of dimensions. In KD tree the parameter space is recursively partitioned along the data axes and data points are filled dividing the space into nested orthotropic regions. Since the partitioning is done only along data axes the construction of KD is very fast and D-dimension is not computed. After construction the nearest neighbor of query point can be determined with $O[\log(N)]$. But with the growth of D this approach becomes inefficient. In scikit-learn KD tree neighbors uses $KDTree$ class for computation and searches are specific to keyword $algorithm = 'kd\_tree'$.

Ball Tree: The ball tree data structure was developed to meet the inefficiencies of KD tree in higher dimensions. Ball tree data structure partitions data in a series of nesting hyper-spheres [22] whereas KD tree did along Cartesian axes. This gives very effective result in very high dimensions but also increases the cost.

In ball tree data is divided recursively into nodes interpreted by centroid C and radius r in a way that each point lies in the hyper-sphere defined by r and C. By triangle inequality $|x + y| \leq |x| + |y|$ the number of points for neighbor search is reduced. With this the single distance between a tests point and the centroid can be calculated which is sufficient to establish a lower and upper bound on the distance to all points within the nodes. The spherical geometry of ball tree nodes surpasses the KD tree in high dimensions even though the performance is mainly dependent on the structure of training data. In scikit-learn the neighbors searches are specific to keyword $algorithm = 'ball\_tree'$ and are computed using class $sklearn.neighbors.BallTree$.

In Ball tree and KD tree the cost of construction in construction phase becomes negligible for many queries. The construction cost take a significant fraction of the total cost for small number of queries. So for small queries brute force is better than tree based method.

If $k < N/2$ the $algorithm = 'auto'$ selects 'kd_tree' and the '$effective\_metric\_$' is in the '$VALID\_METRICS$' list of 'kd_tree'.

If $k < N/2$ it selects '$ball\_tree$' and the '$effective\_metric\_$' is in the '$VALID\_METRICS$' list of '$ball\_tree$'.

If $k < N/2$ it selects '$brute$' and the '$effective\_metric\_$' is not in the '$VALID\_METRICS$' list of kd_tree or $ball\_tree$.

If $k \leq N/2$ it selects '$brute$'. This is done on the assumption of the number of query points same as the number of training points and $leaf\_size$ is close to default value of 30.

Effects of $leaf\_size$: As stated before, brute force search is more efficient than a tree based query for small sample sizes. This can be resolved in ball tree and KD tree by switching to brute force searches internally within leaf node. This can be defined with the parameter $leaf\_size$ which has many effects.

Construction time: In large $leaf\_size$ fewer nodes needs to be created which leads to fast tree construction time.

Query time: Large or small $leaf\_size$ can lead to minimal query cost. If $leaf\_size$ tends to 1 then traversing nodes requires slow query times. If $leaf\_size$ is closer to the size of training set the queries becomes brute force.

Memory: With the increase of $leaf\_size$ the memory to store a tree structure decreases. This is important in case of ball tree as it stores D-dimensional centroid for each node. $1/\ leaf\_size$ is the size of the training set space is needed to store $BallTree$.

$leaf\_size$ is not used in brute force queries.

We used 15 neighbors ($n\_neighbors = 15$) i.e. depending on the values 15 neighbors it gives prediction.

3.4.3 Support Vector Regression

The function of Support Vector Classification can be extended further to solve regression problems. This method is called Support Vector Regression.

The cost function of building a Support Vector Classification model does not count the training points which are beyond specific margin. As a result, the model depends on a subset of the training data. Similarly the cost function of building a Support Vector Regression leaves out any training data close to the model prediction. For which, the model depends on a subset of the training data.

Even though Support Vector Machines are powerful tools, their compute and storage requirements are increasing with the number of training vectors. Scikit-learn has class named $libsvm$ [23] which implements SVR. SVR has a quadratic programming problem (QP) core which separates support vectors from the rest of the training data. $libsvm$-based implementation uses the QP solver scales between $O\left(n_{features} \times {n_{samples}}^2\right)$ and $O\left(n_{features} \times n_{samples}^3\right)$ which depends on the efficiency of $libsvm$ cache (dataset dependent). For very sparse data $n\_features$ should be replaced by the average number of non-zero features in a sample vector.

In scikit-learn the support vector machines support both dense and sparse sample vectors as input. For the efficient performance, C-ordered $numpy.ndarray$ (dense) or $scipy.sparse.csr\_matrix$ (sparse) with data type float64 is preferred.

In the library of scikit-learn three different implementation of SVR is given. They are:

- $\varepsilon - SVR$
- $NuSVR$
- $LinearSVR$

A faster implementation is possible in $LinearSVR$ than $\varepsilon - SVR$ as it only considers linear kernels. We have implement $\varepsilon - SVR$ in our dataset. Mathematical formulation of $\varepsilon - SVR$ is given below:

Assuming a set of training points,$\{(x_1, y_1), \dots, (x_n, y_n)\}$ where x_i is a feature vector and $x_i \in R^n$ and $y_i \in R^1$ is the target output. If the given parameters $C > 0$ and $\varepsilon > 0$ then according to Vapnik [18] Support Vector regression is

$$\min_{w,b,\xi,\xi^*} \frac{1}{2} w^T w + C \sum_{i=0}^{n} \xi_i + C \sum_{i=0}^{n} \xi_i^* \tag{19}$$

$$Subject\ to \quad w^T \phi(x_i) + b - y_i \leq \varepsilon + \xi_i,$$

$$y_i - w^T \phi(x_i) - b \leq \varepsilon + \xi_i,$$

$$\xi_i, \xi_i^* \geq 0, i = 1, \dots, n.$$

The twofold problem is

$$\min_{\alpha,\alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon \sum_{i=0}^{n} (\alpha_i + \alpha_i^*) + \sum_{i=0}^{n} y_i (\alpha - \alpha^*) \tag{20}$$

$$Subject\ to \quad e^T (\alpha - \alpha^*) = 0$$

$$0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \dots, n$$

$$Where\ Q_{ij} = K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)\ \ which\ is\ the\ kernel\ function$$

After solving the above equation, the approximate function is

$$\sum_{i=0}^{n}(-\alpha_i + \alpha_i^*)K(x_i, x) + b \tag{21}$$

In the above equation notations are,

$$w = is\ a\ directed\ vector\ orthogonal\ to\ the\ line\ defined\ by\ w^T x = b$$

$$\xi = slack\ variables\ which\ allow\ for\ violations\ of\ the\ constraints$$

$$C = controls\ the\ trade\ off\ between\ the\ penalty\ and\ margin$$

$$e = [1, \dots, 1]^T$$

In scikit-learn these parameters can be accessed through $dual\_coef\_$ which contains the difference between α_i, α_i^*, $support\_vectors\_$ which holds the support vectors, and $intercept\_$ which holds the independent term b.

The fit method trains the model which takes as argument vectors X, y where y is expected to have floating point values. We used default parameters of library for implementation of SVR where Penalty parameter C=1.0, ε =0.2 (specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value), $degree = 3$ (only for Gaussian Radial Basis function and Polynomial kernel).

3.4.4 Decision Tree Regression with AdaBoost:

$DecisionTreeRegressor$ is a class capable of performing regression on a dataset. Like with other regressions, $DecisionTreeRegressor$ takes as input two arrays. An array X of size $[n_{samples}, n_{features}]$ of the training points, and an array y of integer values having size $n_{samples}$ of the class labels for the training points.

An efficient implementation for the construction of decision tree is offered by scikit-learn. By presorting the features before training and keeping the label count the total cost of the algorithm becomes $O(n_{features} n_{samples} \log(n_{samples}))$. This is optional for all tree based algorithms.

Tree algorithms: In 1986 Ross Quinlan developed ID3 (Iterative Dichotomiser 3). This algorithm creates a multiway tree in which finds the categorical feature for each node that yields the largest information gain for categorical target. These trees are grown as big as possible and then pruned to improve the ability of the tree to generalize to unseen data. C4.5

descendant of ID3. The trained trees are converted to sets of if-then rules by C4.5. The accuracy of each rule assessed to determine the order in which they should be applied. Removing a rule's precondition if the accuracy of the rule improves without it is done by pruning. CART (Classification and Regression Trees) is same as C4.5 but it does not compute rule sets and it supports numerical target variables. Binary trees are constructed using the feature and threshold that yield the largest information gain at each node. Optimized version of CART algorithm is used by scikit-learn.

Mathematical Formulation: Assuming training vectors $x_i \in R^l, i = 1, \dots, n$ and targeted values containing vector $y \in R^n$, partition of the space is made recursively such that the samples with the same labels are in a group.

Suppose the data at node k be represented by P. Splitting is done using $\theta = (j, t_k)$ having of an attribute j and threshold t_m. After partitioning the data is kept into $P_{left}(\theta)$ and $P_{right}(\theta)$ subsets $P_{left}(\theta) = (x, y)|x_j \leq t_k$ and $P_{right}(\theta) = P \setminus P_{left}(\theta)$. Noise at k is calculated using a function $H(\,)$ which calculates impurity. The choice of function depends on the method of solving (classification or regression)[19].

$$G(P, \theta) = \frac{n_{left}}{N_k} H\left(P_{left}(\theta)\right) + \frac{n_{right}}{N_k} H\left(P_{right}(\theta)\right) \tag{22}$$

To minimize impurity parameters are selected using

$$\theta^* = argmin_\theta G(P, \theta) \tag{23}$$

It is continued for subsets $P_{left}(\theta^*)$ and $P_{right}(\theta^*)$ until the maximum depth is reached where $N_k < min_{samples}$ or $N_k = 1$.

The boosting algorithm AdaBoost is included in the module $sklearn.ensemble$ which was introduced by Freund and Schapire in 1995. The main principle of AdaBoost is to fit a sequence of weak leaners (i.e., small decision trees) on data that are modified various times. To produce the final decision the prediction from all are combined by a weighted majority vote. The data is modified at each boosting iteration by applying weights $w_{1,}w_2, w_3, \dots, w_N$ to every training samples. In the beginning the weights are set to $w_i = {}^1/_N$ so that firstly it trains the weak leaner on the original data. In each iteration sample weights are modified one by one and

learning algorithm is reapplied to the reweighted data. At a point the weights are increased of the training examples that were predicted incorrectly by the boosted model at the previous step and the weights are decreased for those were predicted correctly. The examples that are difficult to predict keeps increasing. Every subsequent weak learner is hence forced to concentrate on the examples hat are missed by the previous ones in the sequence.

The parameter $n\_estimators$ controls the number of weak learner. The contribution of the weak leaners in the final combination is controlled $learning\_rate$ parameter. By default, weak learners are decision stumps. Through the $base\_estimator$ parameter different weak learner can be specified. To tune to obtain good results the parameters are $n\_estimators$ and the complexity of the base estimators.

Decision Tree Regression with AdaBoost is a powerful model. For parameters we used $max\_depth = 16$ for Decision Tree Regression. That is, the depth of tree can be maximum 16. For boosting parameter $n\_estimators = 300$ is used. That is, 299 decision tree is compared with a single tree regressor. If we increase this number, the regressor can fit more details.

3.4.5 Root Mean Squared Error

The most frequently used measures of the differences between predicted values by an estimator and the actual values is the Root Mean Square Error (RMSE). The sample standard deviation of the differences between predicted and observed values is represented by the RMSE. When these individual differences are calculated over the data sample used for prediction then they are called residual and when computed out of sample then they are call prediction errors.

The RMSE of an approximation $\widehat{\theta}$ with respect to a real value θ is defined as the square root of the mean square error:

$$RMSE(\hat{\theta}) = \sqrt{MSE(\hat{\theta})} = \sqrt{E\left((\hat{\theta} - \theta)^2\right)} \tag{24}$$

The RMSE of predicted data $\widehat{y_i}$ for survey of i , for variables y_i is calculated for n numbers of cases using the following formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\widehat{y_i} - y_i)^2}{n}} \tag{25}$$

Then RMSE is normalized using:

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}} \tag{26}$$

Scikit-learn has a class to measure mean squared error of a model which is $sklearn.metrics.mean\_squared\_error$. Its takes input of array y_true containing original values and y_pred containing predicted values of a model then gives a floating number value as output which is MSE of the model. Then we used $sqrt()$ function to calculate RMSE as RMSE is square root of MSE.

CHAPTER 04

Experimental Setup and Result Analysis

In this chapter will give details of the hardware, software used in this project. Then we will discuss about the experimental results of every algorithm used to build short term load forecasting in this project. At the end will give a comparison of performance of four machine learning algorithm.

4.1 Details of Hardware and Software

4.1.1 Configuration of Computer

- Processor: AMD FX^{tm}-8300 Eight-Core Processor, ~3.3GHz
- RAM: 16384MB RAM

4.1.2 Programming Languages

- Python 3.5

4.1.3 Editors and Integrated Development Environment

- PyCharm
- Jupyter Notebook

4.1.4 List of Frame Work and Libraries based on Python

- ScikitLearn [24]
- Tensorflow
- Keras [25]
- Pandas
- numPy
- MatplotLib

4.2 Result Analysis

As our goal is to predict the power consumption of the house the next day depending on the power usage of the appliances of the present day. Model is evaluated on test set. The

error result is obtained from test set. For preprocessing we scaled the data using the library function $scale()$. We used $Kfold$ splits for splitting the dataset as it's a time series dataset. Later we trained out model on train set and the error rate is acquired by evaluating the models against our test set. We used all the default settings of the library and few changes in parameters of few algorithms. We used an API out of 3 APIs of the library to measure our trained model's performance. The API scoring parameter contains model-evaluating tools using cross-validation depends on an internal scoring strategy. We used this API to find the MSE of our models then used square root to find out RMSE and we compared the algorithms based on the result of the API.

4.2.1 Nearest Neighbors Regression:

Nearest Neighbors has RMSE of 1.9331727. As our goal is to predict the total power consumption of a house of the next day depending on the usage of the today's power consumption of the appliances, for which we are getting this much higher RMSE value. In Figure 15 the comparison of the real value and the predicted value of the trained model Nearest Neighbors Regression is given.

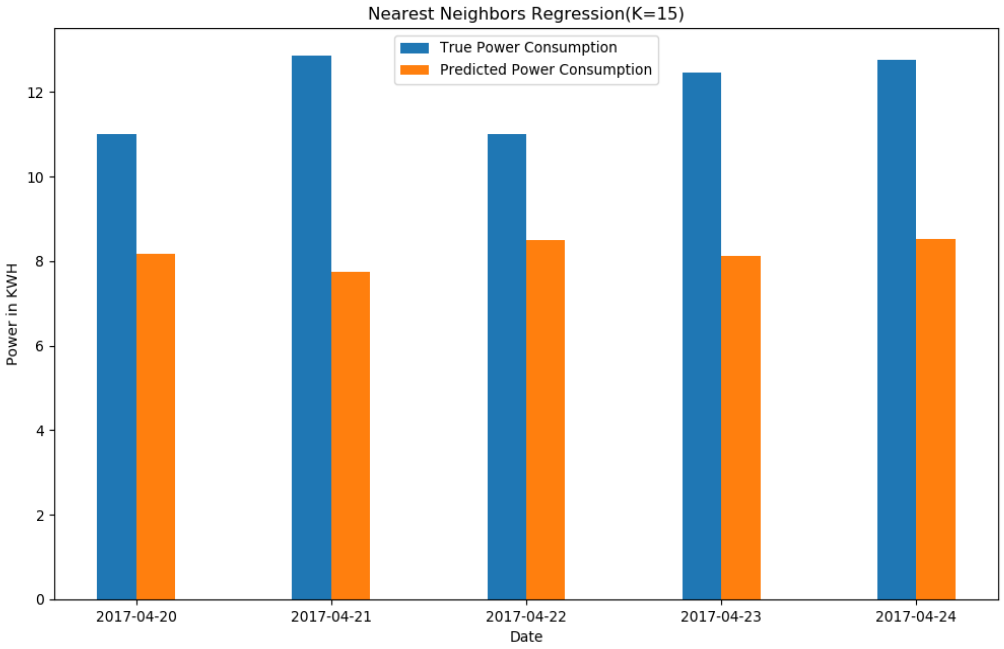

Figure 15. Nearest Neighbors Regression, empirical comparison

4.2.2 Support Vector Regression:

The kernel RBF performs better than other kernels. RBF kernel performs better in this context because of the data. The higher the degree, the performance of other kernels are worse than RBF. SVR using kernel polynomial and Gaussian Radial Basis function with degree of 3 has RMSE of 2.1229618 and 1.8341087 respectively. But kernel linear performs slightly lesser than RBF but better than the polynomial as it has a degree of 1. It has RMSE of 1.8474361. For having the lowest RMSE among the kernel function, we preferred the kernel function RBF for SVR. In Figure 16 the performance of Support Vector regression (kernel RBF) is shown by comparing the true value with predicted value. The predicted power consumption is the output of the model.

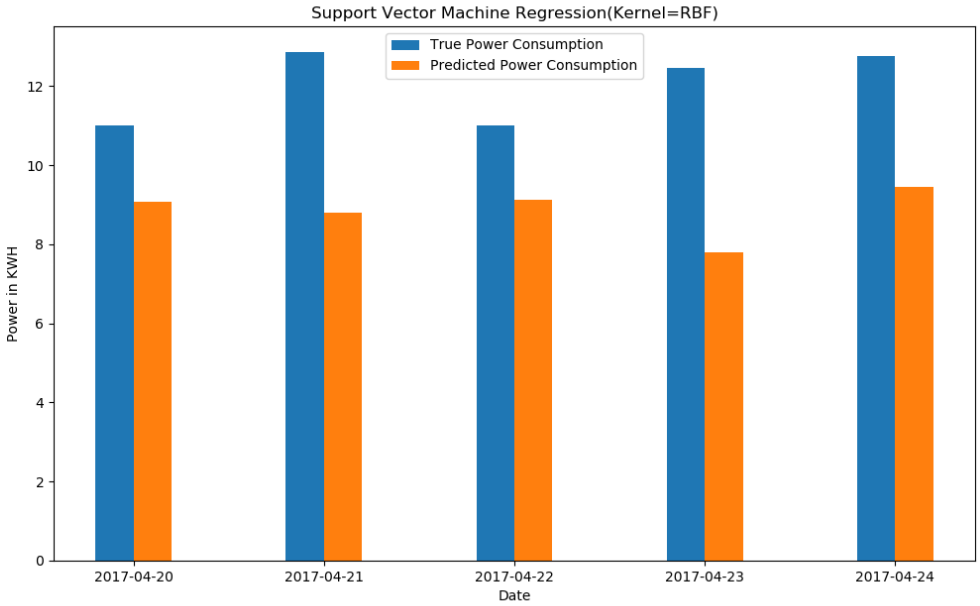

Figure 16. Support Vector Machine Regression, empirical comparison

4.2.3 Decision Tree Regression with AdaBoost:

The RMSE of this model is 1.9202281. As we restricted the depth to 16 and we selected all the features to train the model we are getting much high RMSE. In Figure 17 Evaluation of the model Decision Forest Regression with AdaBoost is shown by comparing predicted and real value.

4.2.4 Long Short-term Memory Network (LSTM)

In experiment we used single cell LSTM network. We have tested with LSTM network with up to 3 LSTM cell stacked top of one another. Stacking more than one LSTM cell made computation heavier but did not give better result. In some cases it went bad.

We have also experimented with length of look back. Here look back is how many samples is given as an input in each time step. We have tested variable length of look back. Most significant were 7 for 7 days, 15 for 15 days, 30 for 1 month. Length of look back between 7 and 15, have given better result than longer look back like 30. We have found best result in look back length 7.

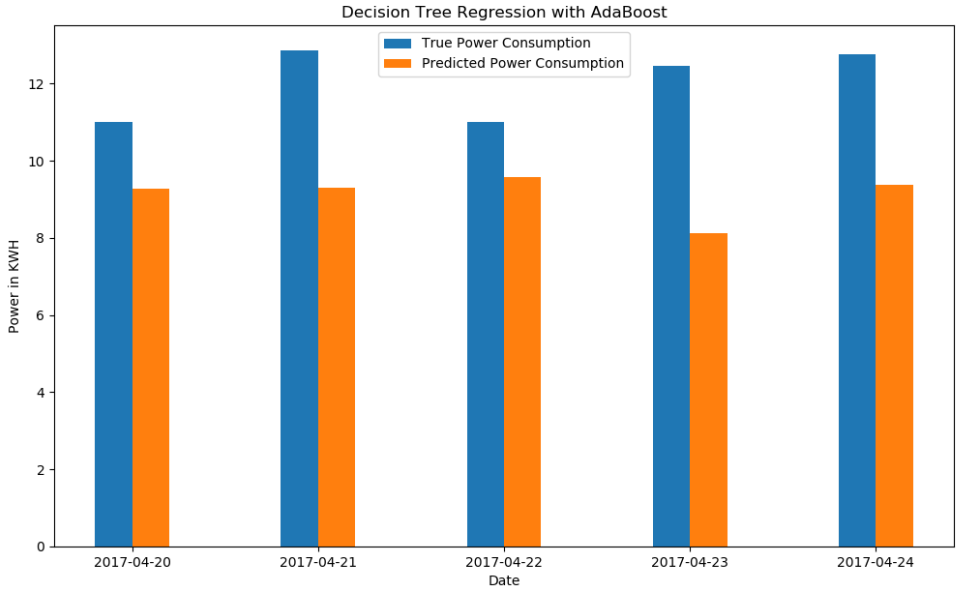

Figure 17. Decision Tree Regression with AdaBoost, empirical comparison

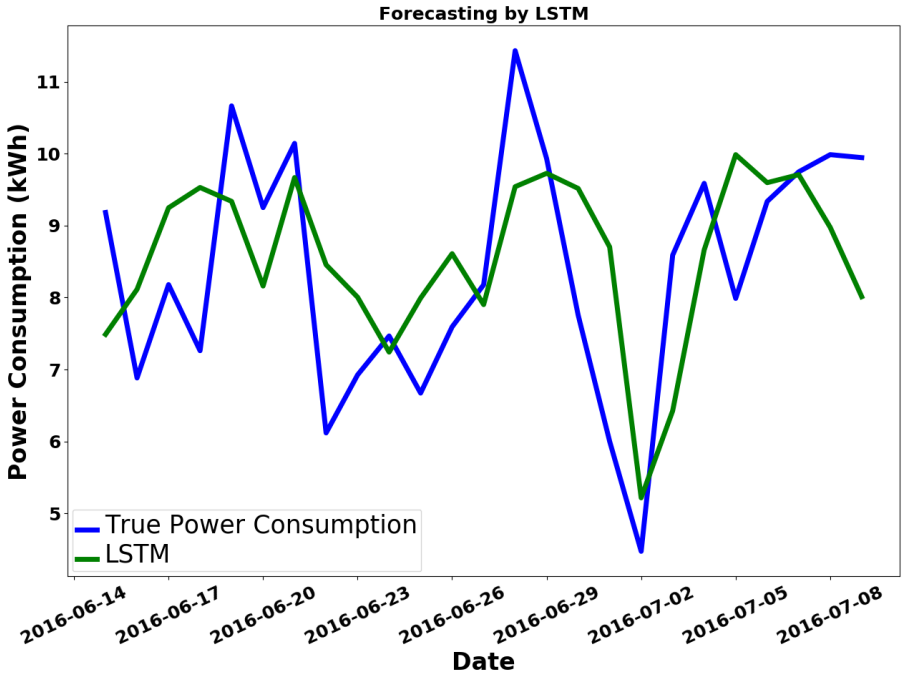

Figure 18. Load forecasting by LSTM

In table we can see LSTM has given lowest RMSE scour. Reason of the lowest scour is ability of a LSTM to process sequence of samples rather than a single sample. In Figure 18 we can see that predicted data points by a LSTM almost catches the pattern of electricity usages. In Figure 19, point by point comparison of 10 test and predicted data point is given.

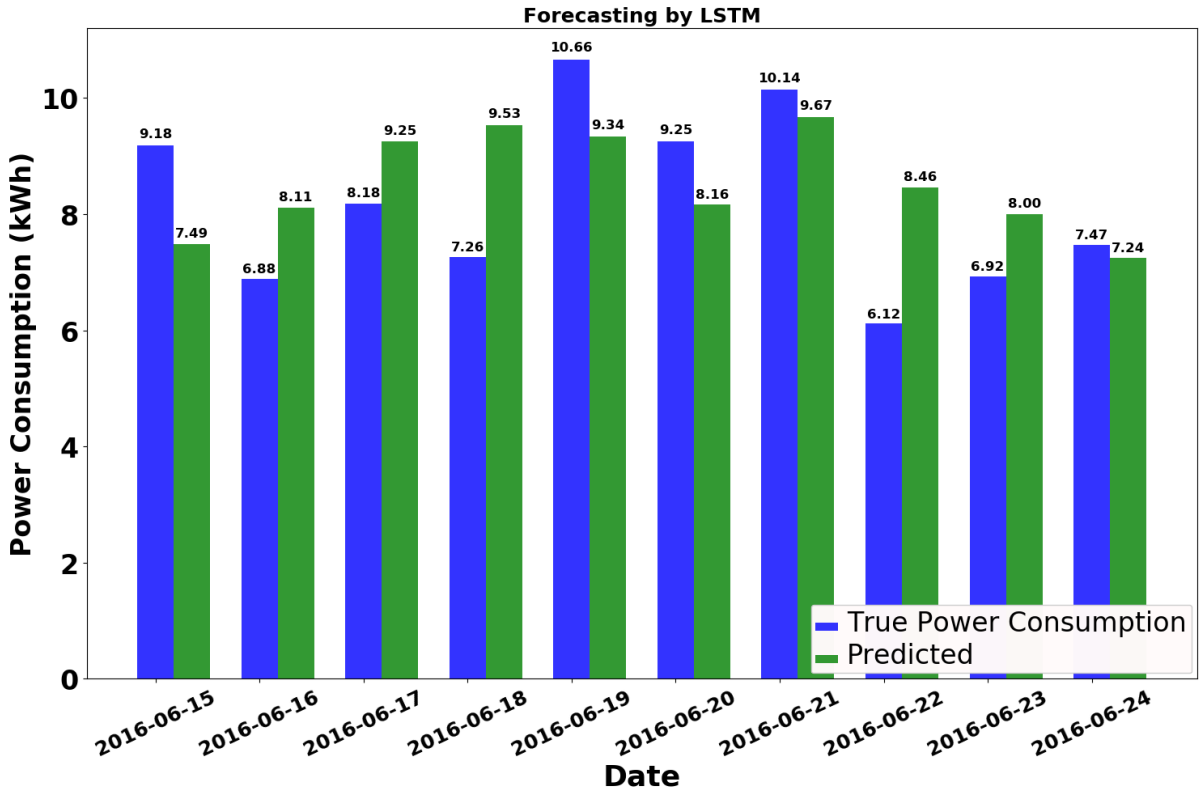

Figure 19. Load forecasting of LSTM, empirical comparison

4.3 Comparison and Result Summary

Table 3. RMSE of ML Algorithms

Algorithms	Error
	Root Mean Squared Error
Nearest Neighbors Regression	1.93
Support Vector Machines Regression	1.83
Decision Forest Regression With AdaBoost	1.86
Long Short Term Memory	1.82

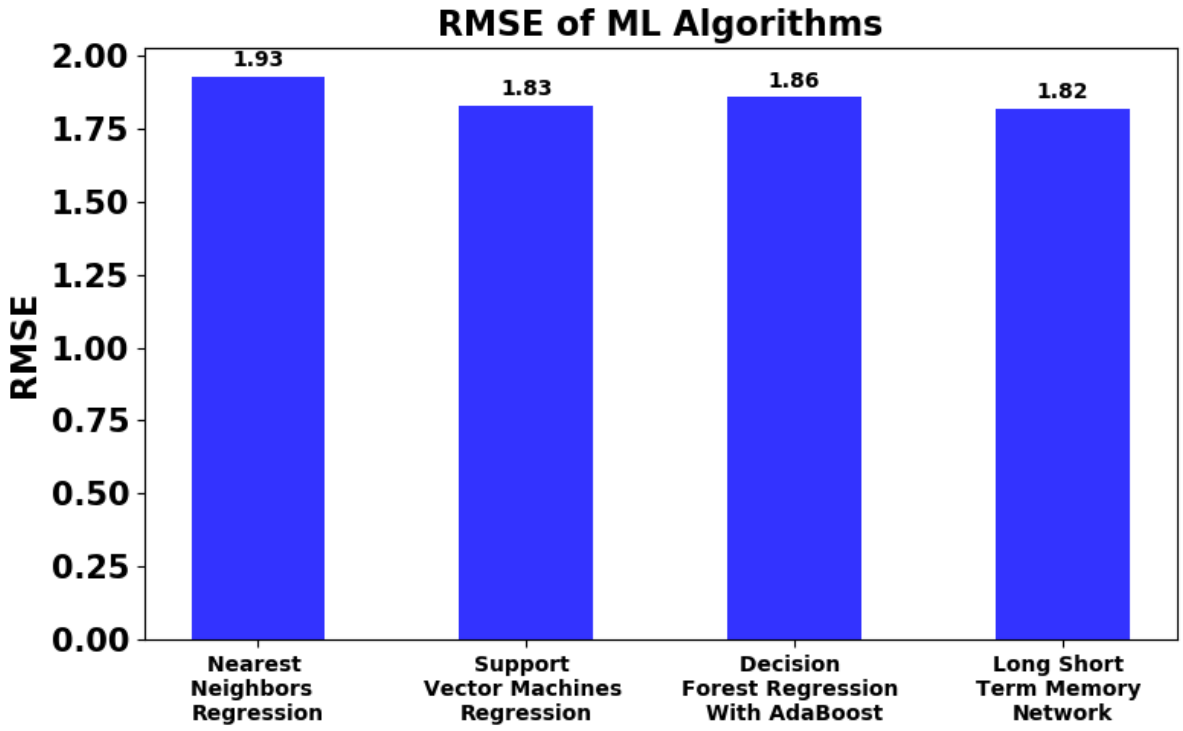

Figure 20. RMSE of ML Algorithms

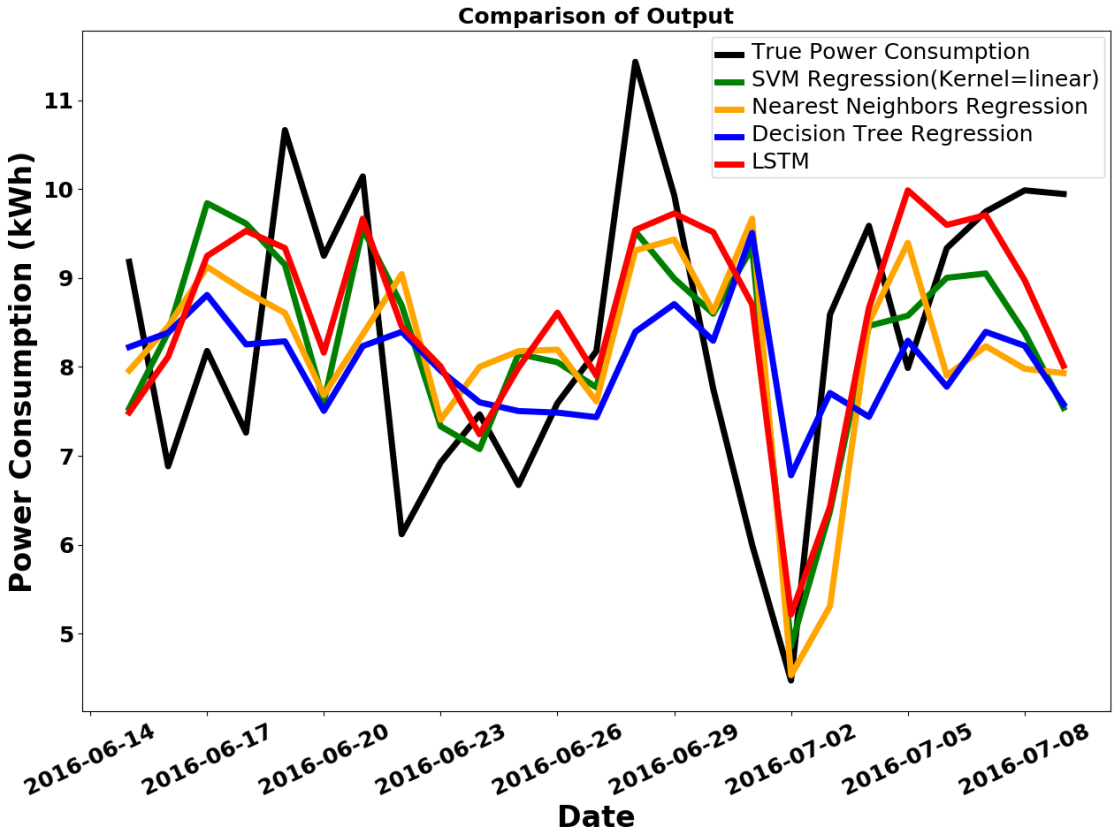

Figure 21. Comparison of four ML Algorithms based on their outputs on same test set

In Figure 21 an overall summary of the outputs of four algorithm is given. Day by day our processors are getting stronger and less power hungry. Also in recent days processors are coming with dedicated core for neural net and artificial intelligence. As a result cost and computational power required for training a neural network will not be problem in the future. Neural networks like LSTM has the ability to adopt with a great variety of patterns and the ability of recognize those pattern. In our experiment LSTM has given better result with compared to Support Vector Machines Regression, Decision Forest Regression with AdaBoost and Nearest Neighbors Regression, Table 3, Figure 20.

CHAPTER 05

Conclusion and Future Work

5.1 Conclusion

In this paper we have presented a system which can give prediction based on data collected from IoT devices. To prove the reliability of the system we have tested the system with real world data sets. We have conducted several experiments to evaluate the performance of four machine learning algorithms and concluded the experiment with a comparison of RMSE loss score. Long Short Term Memory network has given lowest RMSE in the experiment.

5.2 Future Work

The system described in this paper worked on data collected from IoT devices. This system can be implemented for home managements and grid managements. We performed analysis using power consumption data using data clustering based on the time (day). Models can be improved if the data clustering is based on the time interval of hours. This might reduce the error of the models. In future we are looking forward to compare models using different regression based ML algorithms. Privacy is a big concern here. In future we also want to work on the security side of this system. Predictions can be improved selection of features and changing the parameters.

Reference

[1] P. Dempsey, “The Teardown: Google Home personal assistant,” *Eng. Technol.*, vol. 12, no. 3, pp. 80–81, Apr. 2017.

[2] B. Dorsemaine, J.-P. Gaulier, J.-P. Wary, N. Kheir, and P. Urien, “Internet of Things: A Definition & Taxonomy,” in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, 2015, pp. 72–77.

[3] C. Fischer, “Feedback on household electricity consumption: a tool for saving energy?,” *Energy Effic.*, vol. 1, no. 1, pp. 79–104, Feb. 2008.

[4] H. K. Alfares and M. Nazeeruddin, “Electric load forecasting: Literature survey and classification of methods,” *Int. J. Syst. Sci.*, vol. 33, no. 1, pp. 23–34, Jan. 2002.

[5] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu, “Short-Term Residential Load Forecasting Based on Resident Behaviour Learning,” *IEEE Trans. Power Syst.*, vol. 33, no. 1, pp. 1087–1088, Jan. 2018.

[6] G. M. U. Din and A. K. Marnerides, “Short term power load forecasting using Deep Neural Networks,” in *2017 International Conference on Computing, Networking and Communications (ICNC)*, 2017, pp. 594–598.

[7] P. Ray, S. Sen, and A. K. Barisal, “Hybrid methodology for short-term load forecasting,” in *2014 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, 2014, pp. 1–6.

[8] J. Kelly and W. Knottenbelt, “The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes,” *Sci. Data*, vol. 2, p. 150007, Mar. 2015.

[9] P. Louridas and C. Ebert, “Machine Learning,” *IEEE Softw.*, vol. 33, no. 5, pp. 110–115, Sep. 2016.

[10] A. Singh, N. Thakur, and A. Sharma, “A review of supervised machine learning algorithms,” in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 1310–1315.

[11] C. Chatfield, *Time-series forecasting*. Chapman & Hall/CRC, 2001.

[12] “Understanding LSTM Networks -- colah’s blog.” [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 05-Mar-2018].

[13] “Written Memories: Understanding, Deriving and Extending the LSTM - R2RT.” [Online]. Available: https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html. [Accessed: 17-Mar-2018].

[14] A. Graves, “Supervised Sequence Labelling with Recurrent Neural Networks,” p. 18.

[15] F. A. Gers, F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to Forget: Continual Prediction with LSTM,” *NEURAL Comput.*, vol. 12, pp. 2451--2471, 1999.

[16] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[17] N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,” *Am. Stat.*, vol. 46, no. 3, pp. 175–185, Aug. 1992.

[18] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY: Springer New York, 2000.

[19] J. Elith, J. R. Leathwick, and T. Hastie, “A working guide to boosted regression trees,” *J. Anim. Ecol.*, vol. 77, no. 4, pp. 802–813, Jul. 2008.

[20] H. Drucker, “Improving Regressors using Boosting Techniques.”

[21] J. L. Bentley and J. Louis, “Multidimensional binary search trees used for associative searching,” *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[22] S. M. Omohundro and S. M. Omohundro, “Five Balltree Construction Algorithms,” 1989.

[23] C.-C. Chang and C.-J. Lin, “LIBSVM: A Library for Support Vector Machines.”

[24] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[25] F. Chollet and others, “Keras.” 2015.