A comparative study of machine learning algorithms for the analysis and prediction of air quality parameters in Kolkata.

A Dissertation paper

Presented to

The Faculty of the Department of Computer Science

St. Xavier's College, Kolkata

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Science

Supervisor – Dr. Anal Acharya

Submitted by –

Chayan Kumar Sengupta (534)

April, 2020

A COMPARATIVE STUDY OF MACHINE LEARNING ALGORITHMS FOR THE ANALYSIS AND PREDICTION OF AIR QUALITY PARAMETERS IN KOLKATA.

Submitted by –

Chayan Kumar Sengupta (534)

APPROVED:

Dr. Anal Acharya

(Dissertation paper Supervisor)

Romit Beed

(Head of the Department – Dept. of Computer Science)

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled "*A comparative study of machine learning algorithms for the analysis and prediction of air quality parameters in Kolkata*" submitted to Department of Computer Science, ST. XAVIER'S COLLEGE [AUTONOMOUS], KOLKATA, in partial fulfillment of the requirement for the award of the degree of BACHELOR OF SCIENCE (B. SC.) is an original work carried out by Mr……..... [Registration no. ……] under my guidance. The matter embodied in this project is authentic and is genuine work done by the student and has not been submitted whether to this college or to any other institute for the fulfillment of the requirement of any course of study.

……

Signature of the Student

Date : ….....

Name and Address of the Student

……

……

.....

……

Registration no.

……

……

Signature of the Professor

Date: ……

Name, Designation and Address of the Professor

……

……

……

……

ROLES AND RESPONSIBILITIES FORM

Name of the project :

A comparative study of machine learning algorithms for the analysis and prediction of air quality parameters in Kolkata.

Date :.....

We, all the team members admit that, we brought this project come to an end with each one's significant contribution. Each one's dedication, passion for the subject and hard work has been equally important throughout the project.

Name of Team Member	Role	Tasks and Responsibilities
Sagnik Deb	**UI Designer and Team Coordinator**	**GUI Designing, Progress Tracking, Requirements Analysis.**
Reshav Paul	**Data Analyst, System Designer, Model Designer**	**System Coding, Data Pre-processing, Model Improvement analysis**
Chayan Kumar Sengupta	**Backend Designer and Model Designer**	**Backend Coding, Model Coding.**

Name and Signatures of the **Project Team Members**

Sagnik Deb ……. Reshav Paul ……

Chayan Kumar Sengupta .....

Signature of the Professor ……

Date : …….

ACKNOWLEDGEMENTS

This project would not have been possible without the kind help and support of some people. In the very first place, I would like to thank the Department Of Computer Science, St. Xavier's College (Autonomous), Kolkata for giving us this golden opportunity to carry out this dissertation project that has helped us to evolve genuine interest in the subject; study of research papers and has nurtured the spirit of innovation within us.

I would also like to thank our guide, Dr. Anal Acharya, who has always been there to help us out to proceed with the work and continued giving new ideas that would make the project better and more efficient. Without his constant support and encouragement this project might have remained a mere dream. Working under him was indeed a great enriching experience for us. Working together in a team; helping each other in various ways to successfully accomplish the project within stipulated time indeed defined the word team-spirit to its fullest virtue, and improved our decision-taking, leadership and team-work abilities.

Last but not the least; I would like to thank all those people who have directly or indirectly extended their kind hand of help, and their spirit of encouragement in way of completion of the project work.

ABSTRACT

Air quality forecasting is one of the core elements of contemporary Urban Air Quality Management and Information Systems. Time series prediction techniques have been used in many real-world applications such as financial market prediction, electric utility load forecasting, weather and environmental state prediction, and reliability forecasting. Accurate and unbiased estimation of the time series data produced by these systems cannot always be achieved using well known linear techniques, and thus the estimation process requires more advanced time series prediction algorithms. Such systems are usually set up in order to serve environmental legislation needs and are directed towards decision makers (for atmospheric quality problem reduction) and citizens (for early warning and information provision). These forecasting methods that are available does not always lead to forecasting success, as the specific characteristics of each area of interest and the complicated, mostly chaotic relationships between air quality, meteorology, emissions and topography, limit the effectiveness of the methods used. Depending on the data availability we have chosen three air quality parameters (PM_{10}, NO_2, SO_2) and six meteorological parameters (Air temperature, Pressure station level, Relative humidity, Horizontal visibility, Dew point temperature, Mean Wind Speed).We have implemented different machine learning algorithms to comparatively study the trends in each of the air quality parameters and have taken the final models for each parameter based on their optimum performances. The algorithms we have implemented include multivariable linear regression, polynomial regression, support vector regression, decision tree, random forest and neural network models like long short term memory (LSTM) recurrent neural network, gated recurrent unit (GRU), multi-linear perceptron neural network (MLP) and wide deep MLP. Correlation matrix was used to get the better understanding of AQI and its dependency on air quality parameters. We have also designed a web application where a user can choose a date and our model will give an accurate prediction of that particular day's PM_{10}, NO_2, SO_2 and AQI values and a message regarding health issues.

TABLE OF CONTENTS

Chapter I

Introduction

1.1 Purpose of the project

Kolkata, capital of West Bengal has many environmental issues which affects it's biophysical environment. According to data collected in 2010 by the Central Pollution Control Board (CPCB), Kolkata, is among the worst affected cities when the issue is Air Pollution. Kolkata faces the degradation of Air Quality especially during the winter season and hence this project aims to build a proper prediction system analyzing all the affecting meteorological parameters and also provides with health advisories which I believe will work for the betterment of the city.

1.2 Problem statement

The idea is to analyze and predict the ambient air quality standard (which is measured with the help of air quality index), air pollution parameters (PM_{10}, NO_2, SO_2) of Kolkata and provide with its impact on health, some conclusive statements from observations in near future.

1.3 Problem definition and approach

Air quality is measured with the help of AQI (Air Quality Index). Depending on this value of AQI, the air quality can be classified as hazardous, unhealthy, good etc. AQI is calculated on a daily basis taking the maximum value from the sub index values of each air pollutant calculated independently. Depending on the data availability we have chosen three air quality parameters (PM_{10} , NO_2 , SO_2) and six meteorological parameters (Air temperature, Pressure station level, Relative humidity, Horizontal visibility, Dew point temperature, Mean Wind Speed).We have implemented different machine learning algorithms to comparatively study the trends in each of these air quality parameters and have taken the final models for each parameter based on their optimum performances. The meteorological parameters are our input/independent variables and air quality parameters are our output/dependent variables. As we are dealing with continuous data, our problem is a regression problem. The algorithms we have implemented include multivariable linear regression, polynomial regression, support vector regression, decision tree, random forest and neural network models like LSTM, GRU, MLP and wide deep MLP. The models which have performed well are support vector regression, random forest and neural network models. We have chosen four metrics to evaluate our models which are mean absolute

error, mean squared error, root mean squared error and r2 score. We have also calculated the accuracy of a model by finding out the percentage of predictions having less than 20% error from a series of predictions on the test data. To get better accuracy we have introduced some artificial parameters like Day no. (1-365 or 366), previous day's parameter values etc. Correlation matrix was used to get the better understanding of AQI and its dependency on air quality parameters. Examining different plots and the correlation matrix, we have found that AQI value is most heavily affected by PM_{10} , followed by NO_2 and then SO_2 . We have examined the air quality trends from the plots to arrive at some predictive conclusions like which parameter will have less impact in near future etc. We decided to analyze the application of each model to the prediction parameters separately which meant that we could alter our chosen set of input parameters for each model we create to achieve optimal performance on the test set. For this we tried different combinations of parameters starting with all of them and then progressively removing the ones that did not provide much performance gains. However, in some cases, although rarely there were some parameters whose individual removals did not affect the result much but when removed together, a significant performance drop was noticed. This showed that in some models the parameters were influencing the result as a cluster and not independently. This further supports the fact that air pollution levels depend on a group of parameters not necessarily in a one to one relation. Besides the input parameters specific models had other parameters that were tested on many different values and the most optimum were chosen. Such examples include choosing a degree of 2 for polynomial regression, the 'rbf' kernel for support vector regression, 30-50 trees for random forests and neural network hyperparameters. The data set was divided into training and testing sets evenly with the first 80% of records allocated to the training set. We chose this approach over random selection because of the time dependent nature of the problem. During training we tested both and found that our selected partition was performing better than random splits. Some specific models also needed the data to be scaled so that each the effect of all the parameters were equally represented. The models were tested on the test set and their performance was evaluated by a collection of standard procedures and evaluation metrics the details of which can be found under the Implementation and Testing section, as well graphs plotted using the predicted and actual values.

Chapter II

Analysis

2.1 Solution

A study of similar papers indicates that the air quality parameters in a location primarily depend on the meteorological parameters in that location and recent values of the corresponding parameters. A comparative study of ML algorithms required us to collect data for a significant number of meteorological parameters along with air quality parameters PM_{10}, NO_2, SO_2. The target timeframe for the data was 2010-2019. However, we could only collect data from **2013 to 2019**.

2.1.1 Data collection

We collected the data for pollution parameters from the official website of West Bengal Pollution Control Board (WBPCB), India. The website has a publicly accessible API and we had written a python script which automatically uses this API to fetch data and store them. Data was collected for the parameters PM_{10}, NO_2, SO_2, $PM_{2.5}$. For meteorological data there was no free source of data from the Indian Govt. websites for the timeframe we were targeting. So, we used a Russian website (rp5.ru/Weather_archive_in_Kolkata,_Dum_Dum_(airport)), which provides weather forecasts and has archived data going back to 2013. The data had a significant number of parameters and was free to download. The data present is for the Dum Dum Airport region in Kolkata. The parameters selected were **Air Temperature, Dew Point Temperature, Pressure Station Level, Relative Humidity, Horizontal Visibility, Mean Wind Speed**.

2.1.2 Dataset preparation

The data for the meteorological parameters didn't have any invalid entries for any record. However, the data for air quality parameters had many entries missing especially for $PM_{2.5}$. So, even though $PM_{2.5}$ has heavy impact on the air quality of Kolkata we had to exclude it from our list of prediction parameters. **PM_{10}, NO_2 and SO_2** also had missing values but they were few in number and were replaced by the average of the previous and next values. The two datasets were joined and we prepared visualization charts with the help of this data. From the visualization we came up with a few more derived parameters. Following are these parameters with an explanation of why we thought they were important-

Day No. – This parameter has values from 1 to 356 (356 for leap years) representing the number of the day in a particular year. We chose this parameter to quantify the date field as PM_{10} and NO_2 appeared to be varying in a similar way over the course of a year.

Year – NO_2 concentration appears to be decreasing every year so we included Year along with Day No. to reflect this trend.

D-1 PM_{10}, D-1 NO_2, D-1 SO_2 – represent corresponding values for the previous day. These parameters are important to reflect the effect of past values on future values.

2.1.3 Model Selection

The type of problem at hand is a **regression problem** based on time series data. Thus, we studied the most used ML models for regression problems and decided to perform a comparative study of the effectiveness of different models with respect to our problem. Some of the most common models we decided to implement were **Multivariable Linear Regression, Polynomial Regression, Support Vector Regression, Random Forest and four variations of Artificial Neural Networks which are the Multi-Layer Perceptron, a wide-deep variation of MLP, and two variations of Recurrent Neural Networks, the Long Short-Term Memory and the Gated Recurrent Unit**. The particular neural networks we worked with are Multi-Layer Perceptron, Wide Deep Network which is a variation of multi-layer perceptron, Long Short-Term Memory and Gated Recurrent Unit which are both a type of Recurrent Neural Network.

2.2 Software requirements

A python environment with the following installed libraries was used for coding the ML models-

- Scikit-learn == 0.20.3
- NumPy == 1.18.1
- Pandas == 1.0.1
- Keras-Applications == 1.0.8
- Keras-Preprocessing == 1.1.0
- Tensorflow==2.0.0

The following libraries were used for data and result visualization-

- Matplotlib == 3.1.3
- Seaborn == 0.9.0

We used Django (== 2.2.5) framework for our back-end programing.

The libraries, used to fetch meteorological data and pollution parameters-

- Requests == 2.22.0

- Urllib3 == 1.25.8

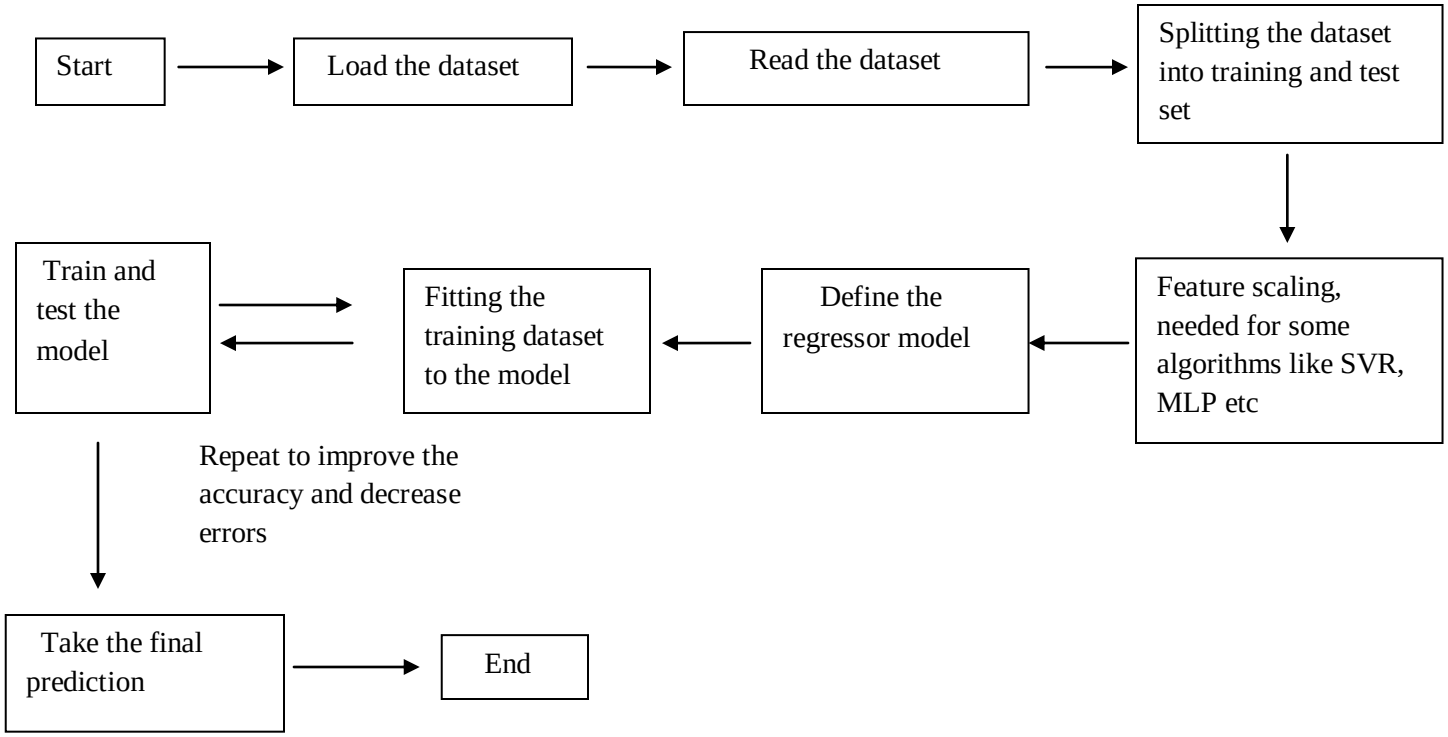
2.3 Hardware requirements

- CPU – Intel Core i3-7100U @2.40GHz
- RAM – 8GB DDR4

Chapter III

Design

3.1 Flow Diagram



Roadmap of the system, Fig. 3.0

3.2 Algorithms

Multivariate Linear Regression

This is quite similar to the simple linear regression model, but with multiple pairs of independent variables and coefficients. In simple linear regression we had only one pair of independent variable and coefficient. So we can say that multivariate linear regression is nothing but the generalization of simple linear regression.

$$Y = B_0 + B_1 * X_1$$

Simple linear regression

Coefficients

$$Y = B_0 + B_1 * X_1 + B_2 * X_2 + \ldots. + B_N * X_N$$

Dependent variable

Constant

Independent variables

Multivariate linear regression

Assumptions of linear regression

Before building a linear regression model we need to check if these are true or not.

- Linearity
- Homoscedasticity
- Multivariate normality
- Independence of errors
- Lack of multicollinearity

Concept of error function

What this model does is, it traverses through all the data points and finds the difference between actual data points and predicted data points and then sums up all these differences. The line passing all those predicted points, which will give minimum sum, will be considered as the best fitted line. Those differences are nothing but the errors.

$$ERROR\_VAL_{line} = SUM(Y_{actual} - Y_{predicted})^2$$

(for all data points of a line)

$$BEST\ FITTED\ LINE = MIN(ERROR\_VAL_{line(i)})$$

for i = 1 to N considering N possible lines

Polynomial regression

This is quite similar to multivariate linear regression but here we try to fit a N^{th} degree curve between independent and dependent variables instead of straight line.

So when we need to apply polynomial regression?

If we have data points like fig 3.2.1 then if we apply simple or multivariable linear regression, we will get best fitted line like fig 3.2.2. But it shows clearly that the line we are getting **under fits** the data points. Here comes the need of polynomial regression. So if we get the curve like fig 3.2.3 this obviously gives better performance over the straight line

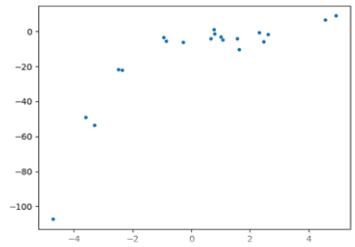


FIG. 3.1

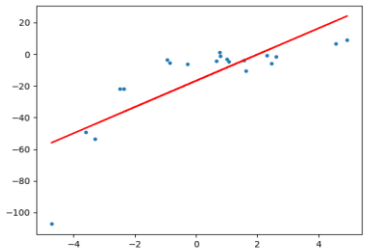


FIG. 3.2

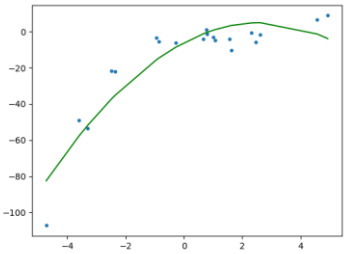
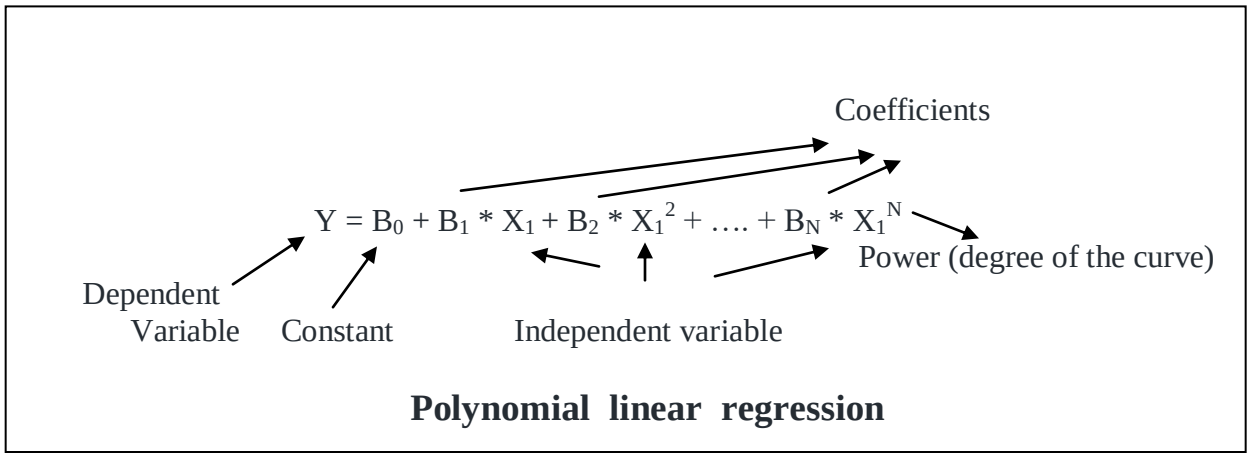


FIG. 3.3

So depending on the value of N, we get quadratic (for N = 2), cubic (for N = 3) curves for fitting. But we also need to take care of **over-fitting** here. For large values of N we might end up getting a model where the curve will pass through most of the data points on training data but will fail on unseen data.

The equation which describes the single variable polynomial regression is mentioned below.



Polynomial linear regression

For multiple variables (say for example if we have 2 independent variables) the 2 degree polynomial equation will have two extra terms than multivariate linear regression

$$Y = B_0 + B_1 * X_1 + B_2 * X_2$$

Multivariate linear regression equation

$$Y = B_0 + B_1 * X_1 + B_2 * X_2 + \underbrace{B_3 * X_1 * X_2 + B_4 * X_1^{\,2} + B_5 * X_2^{\,2}}$$

Polynomial linear regression equation → **extra terms**

This is still considered to be **linear model** as the coefficients/weights associated with the features are still linear. However the curve that we are fitting might be **quadratic** or **cubic** in nature.

Concept of error function

For a chosen N to fit a N degree best fitted curve this model traverses through all data points and finds the ERROR_VAL for each such curves and then selects the minimum of all those ERROR_VAL values.

$$ERROR\_VAL_{\,curve} = SUM(\, Y_{actual} - Y_{predicted}\,)^2$$

(for all data points of a curve)

$$BEST\ FITTED\ CURVE = MIN(ERROR\_VAL_{\,curve\ (i)}\,)$$

for i = 1 to N considering N possible curves

Support vector regression

Support vector machines support linear and nonlinear regression that we can refer to as SVR. Instead of trying to fit the largest possible street between two classes while limiting margin violations, SVR tries to fit as many instances as possible on the street while limiting margin violations. The width of the street is controlled by hyper parameter Epsilon. SVR performs linear regression in a higher dimensional space.

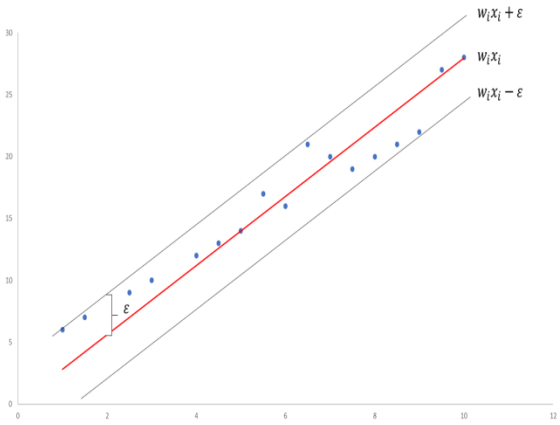


FIG. 3.4

We can think of SVR as if each data point in the training represents it's own dimension.

When you evaluate your kernel between a test point and a point in the training set, the resulting value gives you the coordinate of your test point in that dimension. The vector we get when we evaluate the test point for all points in the training set, k (vector) is the representation of the test point in the higher dimensional space. Once you have that vector you can use it to perform linear regression.

The vectors closest to the test point are referred to as support vectors. We can evaluate our function anywhere so any vectors could be closest to our test evaluation location.

Concept of slack variable

The concept of slack variables is simple: for any value that falls outside of € (**Epsilon)**, we can denote its deviation from the margin as ℑ.

We know that these deviations have the potential to exist, but we would still like to minimize them as much as possible. Thus, we can add these deviations to the objective function.

Minimize :

$$MIN(\left(\frac{1}{2}\right) * ||W||^2 + C \sum_{i=1}^{N} |\Im i|)$$

Constraint :

$$|Y - WiXi| \leq € + |\mathfrak{Ji}|$$

So SVR has a different regression goal compared to linear regression. In linear regression we try to minimize the error between the prediction and data. In SVR our goal is to make sure that our errors don't exceed the threshold.

The optimum degree found is 2 for all the parameters.

Random forest

Random forest is a type of **ensemble learning.** In ensemble learning we use either multiple algorithms or same algorithm multiple times to get better result than before. In random forest we use decision tree as our building block.

If we discuss the algorithm step-wise there are just four steps to perform a random forest regression algorithm.

Step -1 Pick at random K data points from the Training set.

Step -2 Build the Decision tree associated to these K data points.

Step -3 Choose the number Ntree of trees, you want to build and repeat Step -1 and Step -2

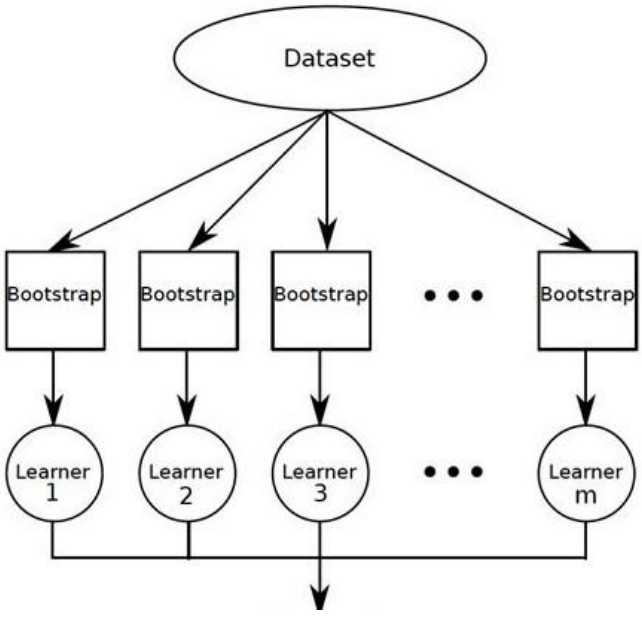
Step -4 For a new data point, make each one of your Ntree trees predict the value of the data point in question and assign the new data point, the average across all of the predicted values.

This improves the accuracy and stability of the final prediction.

Concept of Bagging

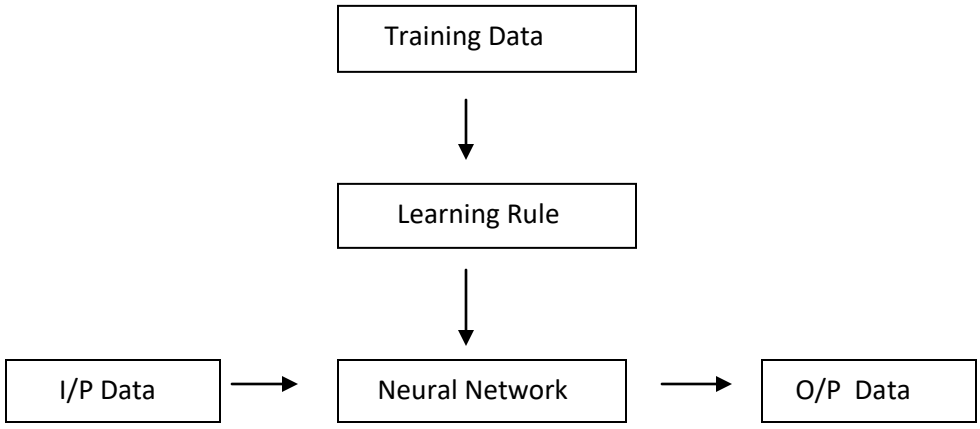
Random forest uses a technique called **Bootstrap Aggregation**, commonly known as **bagging**. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement**.**

The main idea behind this is to combine more than one decision trees rather than relying on a individual decision trees.



Bootstrap aggregation, FIG. 3.5

Neural network algorithms

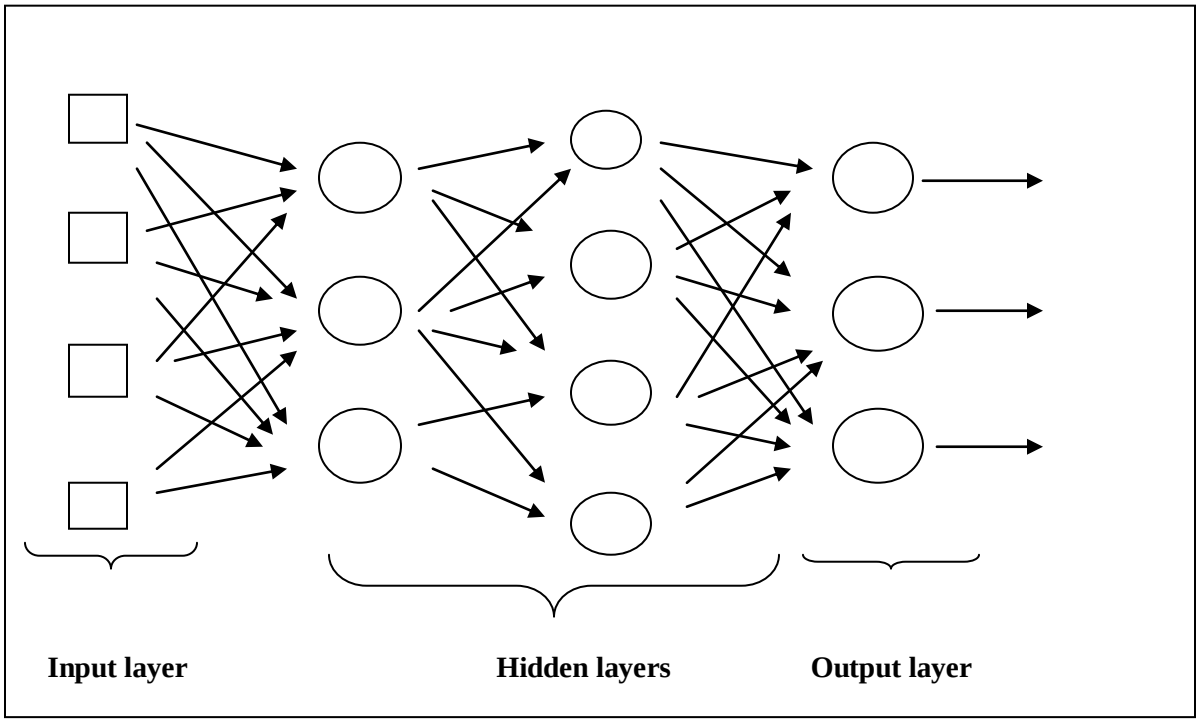


Neural network algorithm intuition, FIG. 3.6

We can implement machine learning model in various ways. Neural network is one of them. In neural network the process of determining the neural network is called **learning rule**. Neural network imitates the mechanism of brain to store information. Like brain neural network has no memory for storing info. In brain neurons transmit signals among themselves to make an association which forms specific information. What is neuron in brain, is **node** in a neural network. That's why brain is called the **gigantic network of neurons**. Neural networks use **connection weights** of nodes.

Multilayer perceptron neural network

The most common type of connection in neural network is **layered connection.** In Multilayer perceptron network, we have multiple layers of nodes. The first layer or input layer transmits the input signal only. This signal passes through hidden layers and reach the output layer. The nodes in hidden layer and output layer accepts the weighted sum from previous node applies a function on this value and passes to the next layer node.



Multilayer perceptron neural network, FIG. 3.7

This function is called the **activation function.** In our case we have used **rectifier activation function** which had better performance than others. The weighted sum is calculated using this formula

$$W = W_1 * X_1 + W_2 * X_2 + W_3 * X_3 + \ldots\ldots + Wn * Xn,$$

where Wi's are weights and Xi's are input signals

Each node in hidden layers and output layers receives this sum as input from previous layer node and applies activation function on it. After the first iteration when the output layer nodes generate the output value then this predicted value is checked with the actual value and an error value is calculated and weights of those nodes are adjusted according to this error and the next iteration begins. This each iteration is called an **epoch**. There are 3 diff ways which help in modifying these weights of the nodes. In our case we have used **mini-batching** where the batch

size defines the number of nodes that are being modified at a time after each iteration. In our case we have also faced a problem called **overfitting** which basically means the model has been overly trained with training data and it results in a mismatch of performance between training

data and test data. To avoid overfitting, we have split the dataset into 3 parts training data for training our model, validation data which takes care of overfitting and test data which gives final accurate prediction. Validation set actually helps to mark that critical point after which the model is getting overfitted.

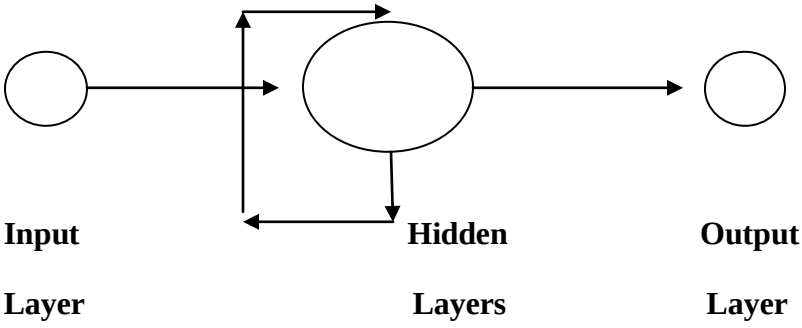
Recurrent Neural Network

Recurrent neural network helps us to deal with **sequence data** more efficiently. **Sequence data** has many forms, one of them is **time series data**. **Time series** data is nothing but sequence data taken in equally spaced time period. In our case as we are dealing with time series data where prediction of a particular date is dependent on the data of previous day's prediction, recurrent neural network is a good option for us. Recurrent neural network uses something called, **sequential memory.** It's something which helps to recognize sequence patterns.

We can say that,

Recurrent neural network = Feed forward neural network + Sequential memory

To implement the concept of sequential memory we add a loop in the feed forward neural network that behaves like a **feedback line** in **sequential circuit.**



Recurrent neural network intuition, FIG. 3.8

But the issue with **recurrent neural network** is it's **short term memory.** Due to **back propagation through time** the **gradient** value which helps the network to learn and perform better by changing weight values, shrinks down and become very small for earlier layers. This is called **vanishing gradient problem**. So they fail to learn from them. This is why **recurrent neural network** faces trouble with **long range dependencies.**

Long Short Term Memory Neural Network

Long short term memory was one of the solutions to the short term memory problem. LSTM uses gates to decide which are the relevant info for the prediction and passes only that info to the next layer. Thus irrelevant info gets left out of the sequence and the rest of the info gets passed down the long chain of layers.

The work flow of LSTM is almost similar like recurrent neural network, only difference is it's **cell state** and various **gates. Cell state** can be considered as the memory for the network which helps to regulate the passing of relevant info down the chain of layers. So It reduces the effect of **short term memory** as it carries info related to earlier states which keeps getting added to the info which is passed to next layer. Gates in LSTM uses **sigmoid activation function (**maps values between 0 and 1**).** A **vector** is formed which contains info related to previous hidden state and input from the current state and is passed to **sigmoid function**. LSTM uses three gates which are **Forget gate**, **Input gate, Output gate** to perform task. **Forget gate** is responsible for deciding the relevance of info, which info should be kept and which should be left out. The vector is passed to the **sigmoid function**. As the output ranges between 0 and 1 the value closer to 0 is considered as irrelevant and is thrown out and a value closer to 1 is passed to next layer. **Input gate** helps in updating the **cell state.** The output from the **sigmoid function** and the output from **tanh function (**tanh function also receives that vector and maps the values between -1 to 1**)** is multiplied to decide which info are relevant and according to that cell state gets updated. **Output gate** decides which info to be passed to the next hidden state. The result of the multiplication of tanh output and sigmoid output is the **next hidden state** which is combined with **new cell state** and is passed to the next time step.

Gated Recurrent Unit Neural Network

The principle, that GRU follows, is quite similar to LSTM. GRU also uses the mechanism of a **gate** to regulate the workflow through the chain of sequences. The difference is instead of **cell state** it uses **hidden state** for information flowing.

GRU has only two gates unlike LSTM which has three gates and they are **reset gate** and **update gate**.

The purpose of **reset gate** is similar to the **forget** and **input gate** of LSTM. It decides which information is relevant to carry forward and which should be thrown out so that **short term memory** problem doesn't arise.

$$Y_t = \sigma\,(Q_r \,.\, [M_{t-1}, N_t])$$

The **update gate** decides how much past information should be forgotten from previous steps. This is useful because the model now can eliminate the risk of **vanishing gradient problem.**

$$Z_t = \sigma\,(Q_z \,.\, [M_{t-1}, N_t])$$

If carefully trained they can perform extremely well even in complex scenarios.

Wide Deep Multi-Layer Perceptron Neural Networks

Wide deep neural network joins two methods of memorizing and generalizing the learning. This associates memory element with it.

It uses wide linear model for memorization alongside a deep neural network for generalization to get the strengths of both. The deep neural network is a feed forward network with large number of hidden layers.

Concept of embedding vector

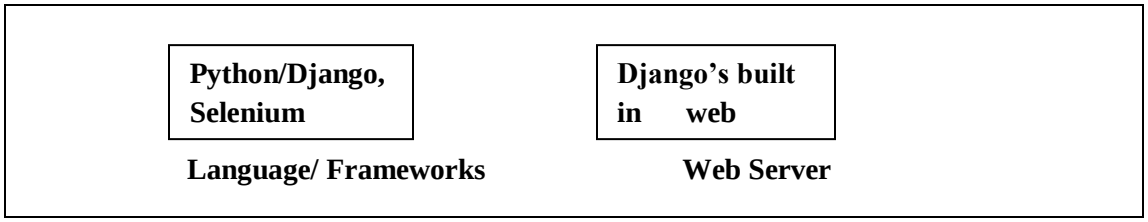
The high dimensional features are converted to lower dimensions and this representation is called **embedding vector.**

This lower dimensional vectors are then combined with the features and then passed as input to the hidden layers. The vectors are randomly initialized. Other parameters are also trained to minimize the cost function

Deep models will generalize better when the embeddings are denser.

3.3 Interface

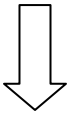
Our Stack

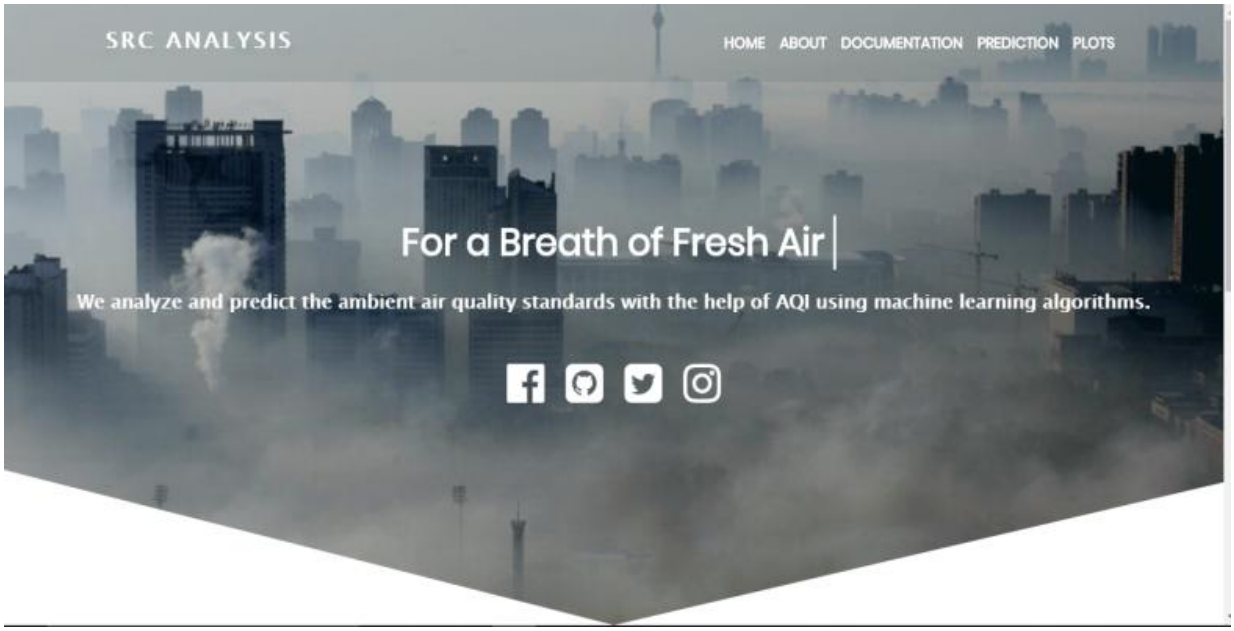


BACK END APPLICATIONS

HTML, CSS, BOOTSTRAP, JAVASCRIPT, Some jQuery Plugins

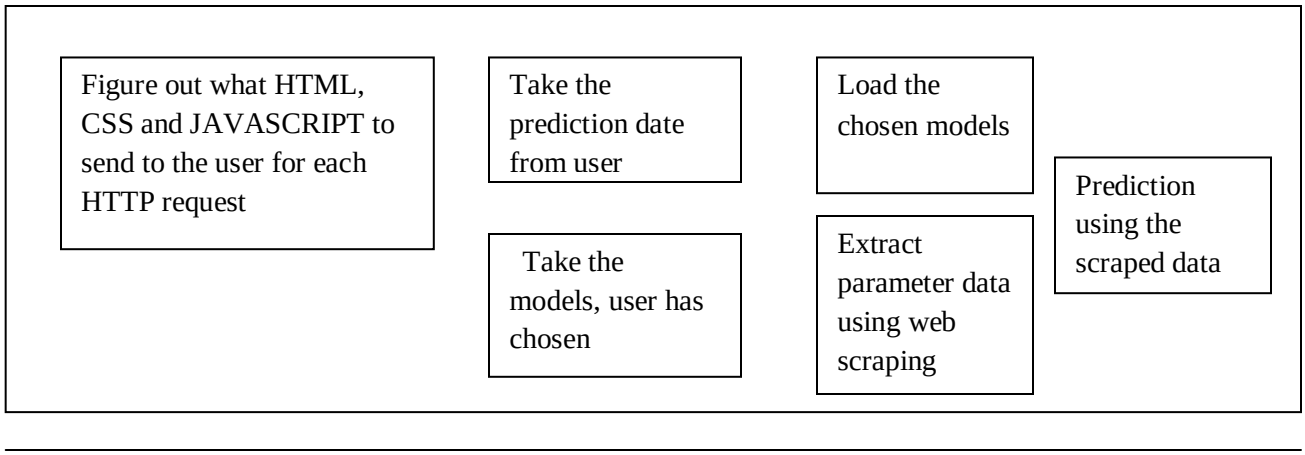
FRONT END APPLICATIONS



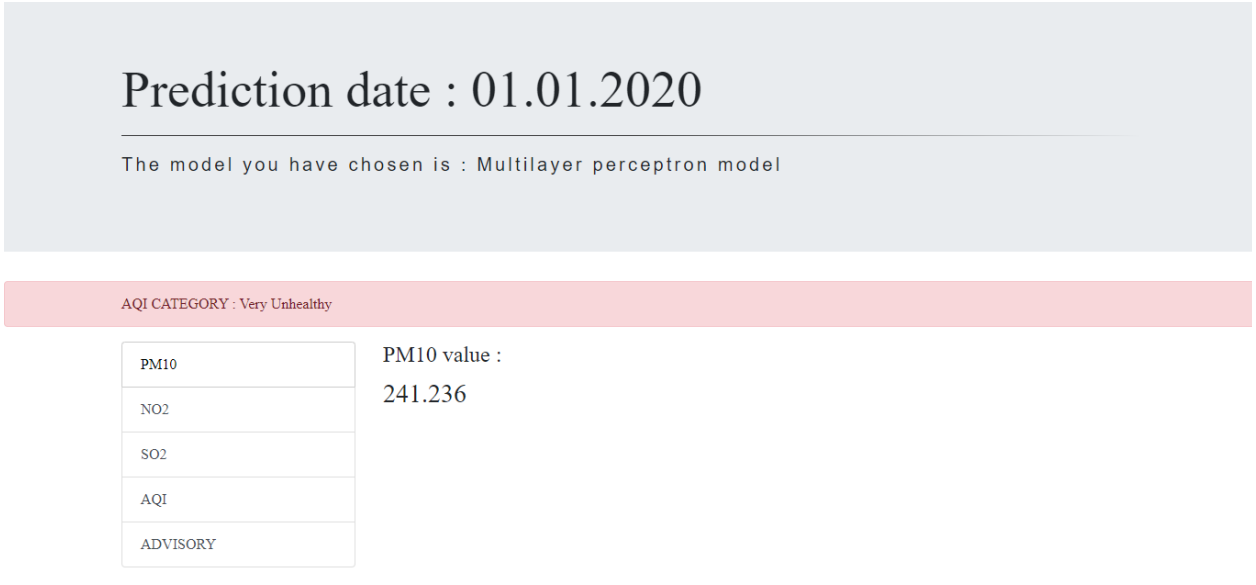


Stack of our system, FIG. 3.9

Backend features



HTML, CSS, BOOTSTRAP, JAVASCRIPT, Some jQuery Plugins



Available backend features to an user, FIG. 3.10

Chapter IV

Implementation and testing

4.1 Preprocessing and model training

4.1.1 Preprocessing

Splitting the data into test and train sets –

The entire dataset contains 2372 entries and an approximate of 80% of the total records were used for training the models and the rest were used for testing. The splitting was not done randomly since we are dealing with time series data. The data was split into two blocks that contained sequential or continuous data.

4.1.2 Model Training

Training machine learning models follows the same general procedure for almost all models. We create a model by using the classes in libraries like sklearn and keras. We create an object of the appropriate class which represents our model. We pass the parameters that define the model which is specific to each model while creating the object. Neural networks also require the data to be scaled to fit between 0 and 1. We then train the model with the training data. After the model is trained, we pass it the values from the test data to get the predictions. We compare the predicted values with the actual values from the test set to find the error.

4.2 Model evaluation

Errors are of different types and they show the effectiveness of the models in making the predictions. We used the following errors to evaluate our models-

- Root Mean Square Error
- Mean Square Error
- Mean Absolute Error

We used two additional metrics to evaluate our models –

- R2 score
- Accuracy – which we define as the percentage of predictions from a series of predictions that have an error of less than 20%.

The models were tested on the above criteria using the predictions obtained from the test set.

4.3 Effective parameters list

Generally, when it comes to multivariate linear regression, we don't just blindly consider all the independent variables at a time and start minimizing the error function. There are mainly two reasons behind this.

i. **Garbage-in, Garbage-out :** The model having so many unnecessary independent variables is considered as garbage model.
ii. **Complexity in explanation :** More you have variables more will be the complexity of the model.

So we should only take into consideration those variables which are best possible independent variables that contribute well to the dependent variable. For this, we go on and construct a correlation matrix for all the independent variables and the dependent variable from the observed data. The correlation value gives us an idea about which variable is significant and by what factor. From this matrix we pick independent variables in decreasing order of correlation value and run the regression model to estimate the coefficients by minimizing the error function. We stop when there is no prominent improvement in the estimation function by inclusion of the next independent feature. This method can still get complicated when there are large numbers of independent features that have significant contribution in deciding our dependent variable.

In our case considering the evaluation matrices and accuracy, we have found these parameters having effective impacts on each pollutants. So this eliminates those two properties mentioned above and makes our model perform well.

Multivariable Linear Regression

PM_{10}	'Previous day's PM_{10} value', 'Dew point Temperature', 'Wind speed', 'Day No.' (artificial variable considering value 1 for 1st January), , 'Pressure station level'

NO_2	'Previous day's NO_2 value', 'Air temperature' 'Horizontal Visibility', 'Relative Humidity', 'Dew point Temperature'
SO_2	'Previous day's SO_2 value', 'Air temperature', 'Day before previous day's SO_2 value', 'Dew point temperature', 'Relative humidity', 'Wind speed'

Multivariable linear regression effective parameter list, TABLE. 4.0

Polynomial Regression

PM_{10}	'Previous day's PM_{10} value', 'Day No.' (artificial variable considering value 1 for 1st January), 'Air Temperature', 'Horizontal Visibility', 'Dew Point Temperature', 'Year'
NO_2	'Previous day's NO_2 value', 'Horizontal Visibility', 'Relative Humidity', 'Pressure Station Level', 'Day before previous day's NO_2 value'
SO_2	'Previous day's SO_2 value', 'Day No.' (artificial variable considering value 1 for 1st January), 'Day before previous day's SO_2 value', 'Year' (year value of the prediction date)

Polynomial regression effective parameter list, TABLE. 4.1

Support Vector Regression

PM_{10}	'Previous day's PM_{10} value', 'Day before previous day's PM_{10} value', 'Air Temperature', 'Horizontal Visibility', 'Dew Point Temperature', 'Wind Speed', 'Relative Humidity'
NO_2	'Previous day's NO_2 value ', 'Pressure Station Level', 'Dew Point Temperature', 'Relative Humidity', 'Day No.' (artificial variable considering value 1 for 1st January), 'Day before previous day's NO_2 value', 'Horizontal Visibility'
SO_2	'Previous day's SO_2 value', 'Day No.' (artificial variable considering value 1 for 1st January), 'Day before previous day's SO_2 value', 'Pressure Station Level', 'Relative Humidity'

Support vector regression effective parameter list, TABLE. 4.2

Random Forest

PM_{10}	'Previous day's PM_{10} value', 'Day No." (artificial variable considering value 1 for 1st January), 'Air Temperature', 'Dew Point Temperature', 'Year', 'Pressure Station Level', 'Horizontal Visibility', 'Wind Speed', The optimum number of trees chosen - 50
NO_2	'Previous day's NO_2 value', 'Day No.', 'Air Temperature', 'Pressure Station Level','Relative Humidity', 'Horizontal Visibility', 'Year', 'Day before previous day's NO_2 value', The optimum number of trees chosen - 30
SO_2	'Previous day's SO_2 value', 'Day No.', 'Air Temperature', 'Pressure Station Level', 'Year', 'Day before previous day's SO_2 value', The optimum number of trees chosen - 30

Random forest regression effective parameter list, TABLE. 4.3

Multi-Layer Perceptron

PM_{10}	'Air Temperature', 'Relative Humidity', 'Horizontal Visibility', 'Year', 'Day before previous day's PM_{10} value', 'Day No." (artificial variable considering value 1 for 1st January), 'Previous day's PM_{10} value' model structure :Three hidden layers having 75 neurons for each, activation function : rectifier(relu), epoch value: 200, optimizer function : lbfgs, learning rate : 0.003 and batch size : 28)
NO_2	'NO_2', 'D-1 NO_2', 'Dew Point Temperature', 'Horizontal Visibility', 'Relative Humidity', 'Year', 'Wind Speed', hidden layers: 30(relu), 30(relu), batch size: 28, learning: Adam(0.001)
SO_2	'SO_2', 'D-1 SO_2', 'Dew Point Temperature', 'Relative Humidity' hidden layers: 10(relu), 10(relu) batch size: 32 learning: Adam(0.001)

Multi layer perceptron neural network effective parameter list, TABLE. 4.4

Wide-deep Multi-Layer Perceptron

PM_{10}	**Deep params:** 'Day No.', 'Air Temperature', 'Wind Speed', 'Day before previous day's PM10', 'Relative Humidity', 'PM10', 'Horizontal Visibility', 'Year', **Wide params**: 'D-1 PM10', 'PM10', hidden layers: 30(relu), 30(relu), 30(relu), batch size: 28, learning: Adam(0.001)
NO_2	Deep params: 'Day before previous day's NO_2', 'Previous day's NO2', 'Wind Speed', 'Dew Point Temperature', 'Air Temperature', 'Pressure Station Level', 'Relative Humidity', 'Year',

	Wide params: 'Previous day's NO_2', hidden layers: 30(relu), 30(relu), batch size: 28, learning: Adam(0.001)
SO_2	Deep params: 'SO_2', 'D-1 SO_2', Wide params: 'SO_2', hidden layers: 30(relu), 30(relu), 30(relu), batch size: 28, learning: Adam(0.001)

Wide deep neural network effective parameter list, TABLE. 4.5

Long Short-Term Memory (LSTM) Recurrent Neural Network

PM_{10}	'Previous day's PM_{10} value', 'Air Temperature', 'Day before previous day's PM_{10} value', 'Pressure Station Level', 'Wind Speed', 'Relative Humidity', 'Horizontal Visibility', 'Year', hidden layers: 30(relu), 30(relu), 30(relu), batch size: 28, learning: Adam(0.001)
NO_2	'Previous day's NO_2', 'Day before previous day's NO_2', 'Dew Point Temperature', 'Wind Speed', 'Air Temperature', 'Pressure Station Level', 'Relative Humidity', 'Year', hidden layers: 30(relu), 30(relu), batch size: 28, learning: Adam(0.001)
SO_2	'Previous day's SO_2', 'Day before previous day's SO_2','Air Temperature', 'Pressure Station Level', 'Wind Speed', hidden layers: 30(relu), 30(relu) ,batch size: 28, learning: Adam(0.001)

LSTM neural network effective parameter list, TABLE. 4.6

Gated Recurrent Unit (GRU)

PM_{10}	'Previous day's PM_{10}', 'Air Temperature', 'Pressure Station Level', 'Day before previous day's PM_{10}', 'Wind Speed', 'Relative Humidity', 'Horizontal Visibility', 'Year' hidden layers: 30(relu), 30(relu), 30(relu) batch size: 28 learning: Adam(0.001)
NO_2	'Previous day's NO_2', 'Day before previous day's NO_2', 'Dew Point Temperature', 'Wind Speed', 'Air Temperature', 'Pressure Station Level', 'Relative Humidity', 'Year' hidden layers: 30(relu), 30(relu) batch size: 28 learning: Adam(0.001)
SO_2	'SO_2', 'D-1 SO_2', 'Air Temperature', 'Relative Humidity', 'Wind Speed' hidden layers: 30(relu), 30(relu), 30(relu) batch size: 28 learning: Adam(0.001)

Gated recurrent unit neural network effective parameter list, TABLE. 4.7

4.4 Testing results and accuracy

PM_{10}

Model	Mean Squared Error	Mean Absolute Error	Root Mean Squared Error	R2 Score	Accuracy (< 20% error)

Multi-Linear Regression	511.257	16.528	22.611	0.914	65.61%
Polynomial Regression	369.562	14.276	19.224	0.938	72.15%
Support Vector Regression	475.763	15.473	21.812	0.92	75.1%
Random Forest Regression	490.259	15.805	22.142	0.9176	72.99%
Multi-Layer Perceptron	376.514	13.446	19.404	0.9367	74.73%
Wide deep MLP	384.695	14.474	19.614	0.9356	73.46%
LSTM	386.02	14.769	19.647	0.9354	72.19%
GRU	382.971	14.57	19.57	0.9359	73.67%

PM10 ‘s performance on different models , TABLE. 4.8

NO_2

Model	**Mean Squared Error**	**Mean Absolute Error**	**Root Mean Squared Error**	**R2 Score**	**Accuracy (< 20% error)**
Multi-Linear Regression	34.621	4.472	5.884	0.862	84.81%
Polynomial Regression	31.364	4.104	5.6	0.875	89.87%
Support Vector Regression	31.427	4.143	5.606	0.874	89.03%
Random Forest Regression	35.772	4.301	5.981	0.857	88.39%
Multi-Layer Perceptron	29.612	4.009	5.442	0.883	88.74%
Wide deep MLP	30.469	4.097	5.519	0.879	88.95%
LSTM	31.039	4.001	5.571	0.877	90.87%
GRU	30.381	4.023	5.512	0.88	90.02%

NO2 ‘s performance on different models , TABLE. 4.9

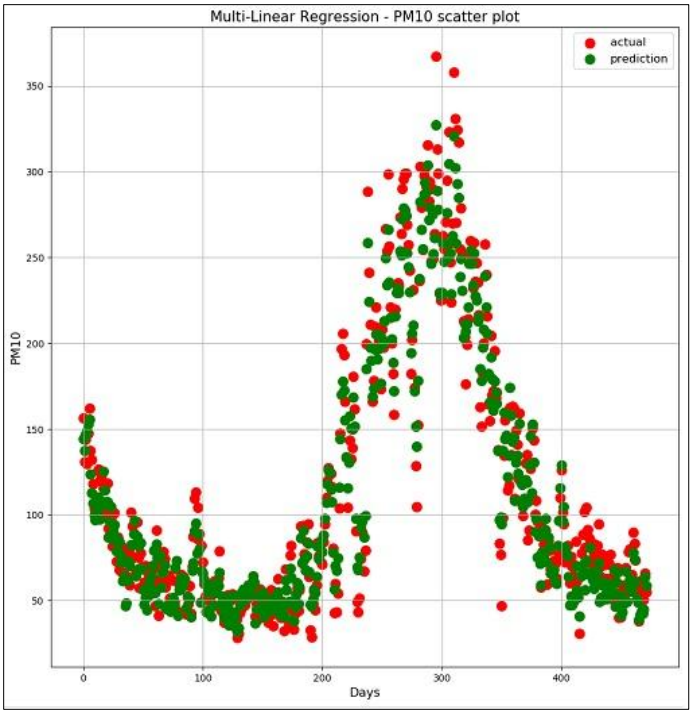
SO_2

Model	Mean Squared Error	Mean Absolute Error	Root Mean Squared Error	R2 Score	Accuracy (< 20% error)
Multi-Linear Regression	2.608	1.299	1.615	0.775	49.58%
Polynomial Regression	2.602	1.174	1.613	0.776	60.55%
Support Vector Regression	2.262	1.102	1.504	0.805	59.49%
Random Forest Regression	3.361	1.365	1.833	0.71	52.11%
Multi-Layer Perceptron	2.527	1.196	1.589	0.784	56.26%
Wide deep MLP	2.67	1.226	1.621	0.775	55.2%
LSTM	2.489	1.191	1.578	0.787	57.32%
GRU	2.466	1.171	1.57	0.789	54.99%

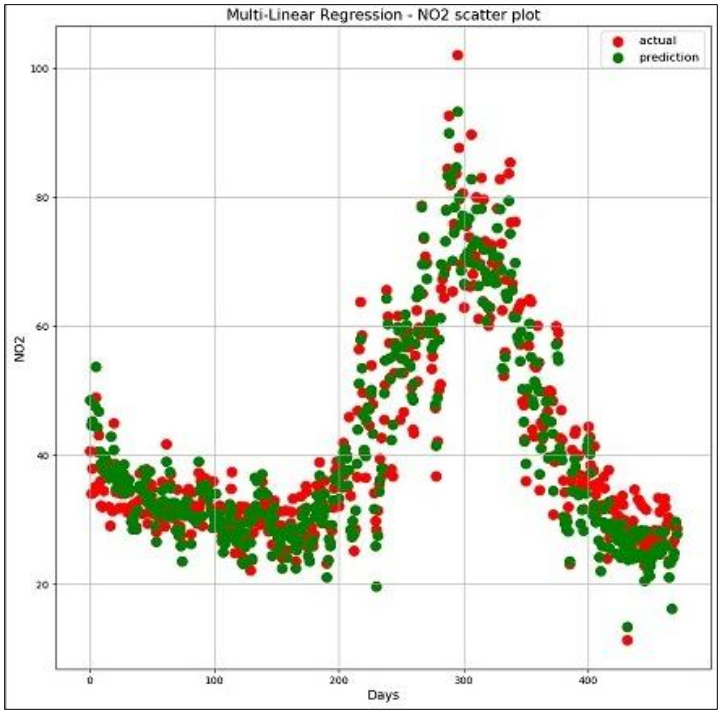
SO2 ‘s performance on different models , TABLE. 4.10

4.5 Plots

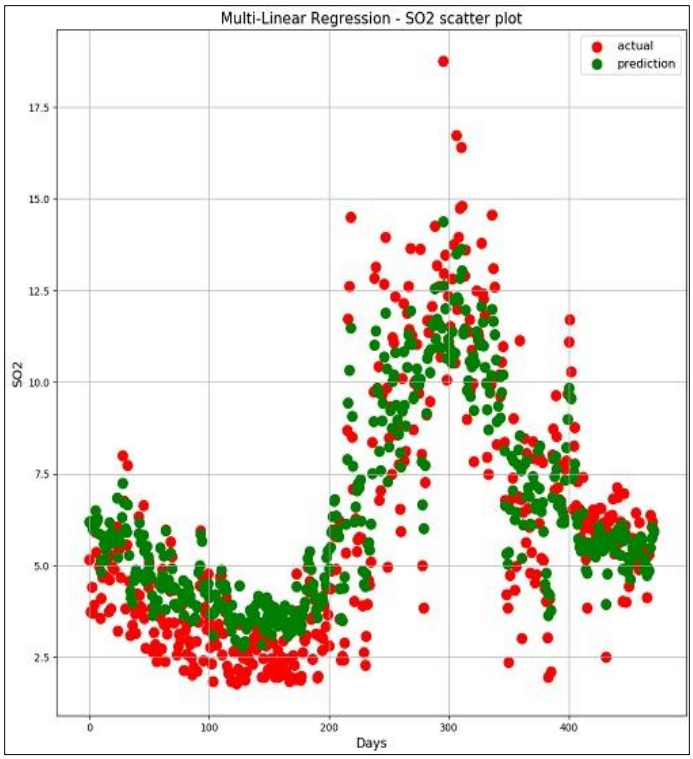
Multivariable Linear Regression



PM10 Multi variable regression, FIG. 4.0

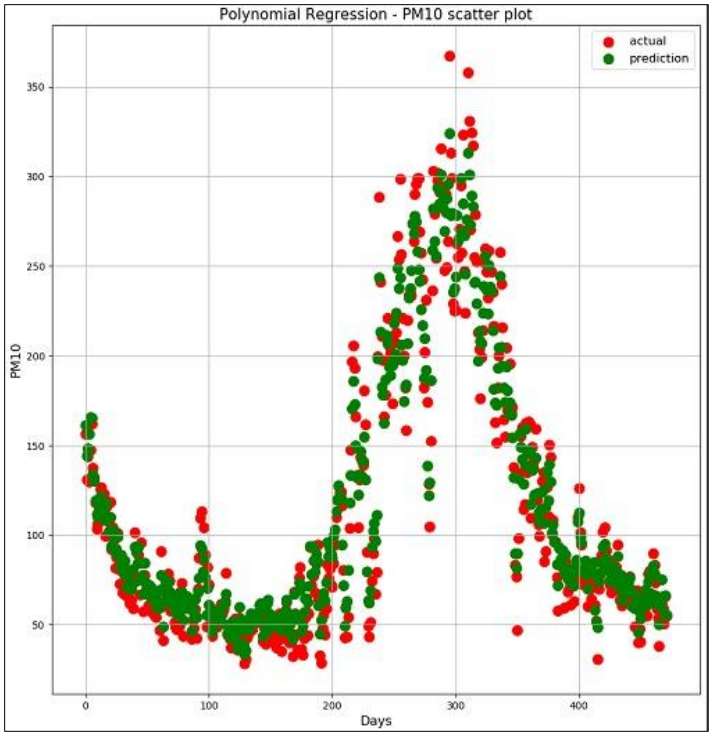


NO2 Multi variable regression, FIG. 4.1

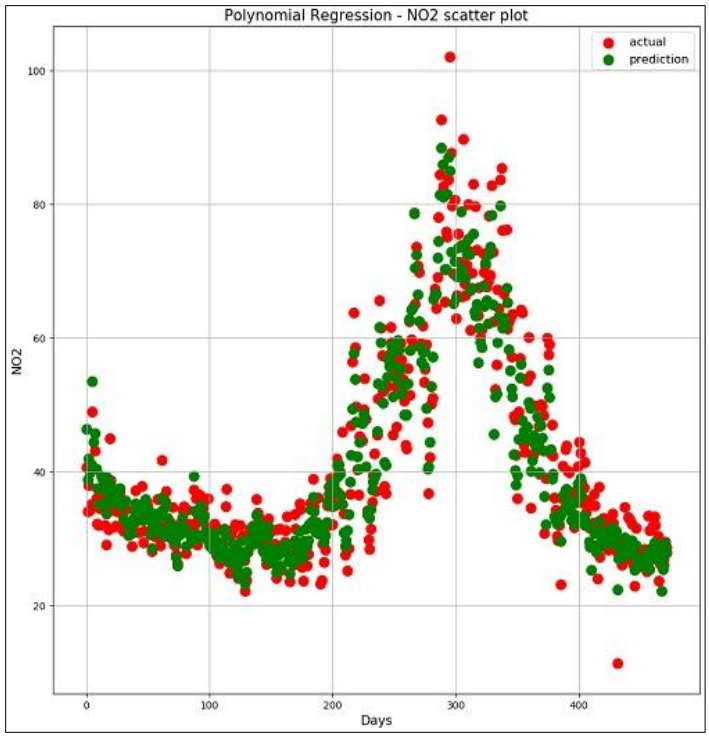


SO2 Multi variable regression, FIG. 4.2

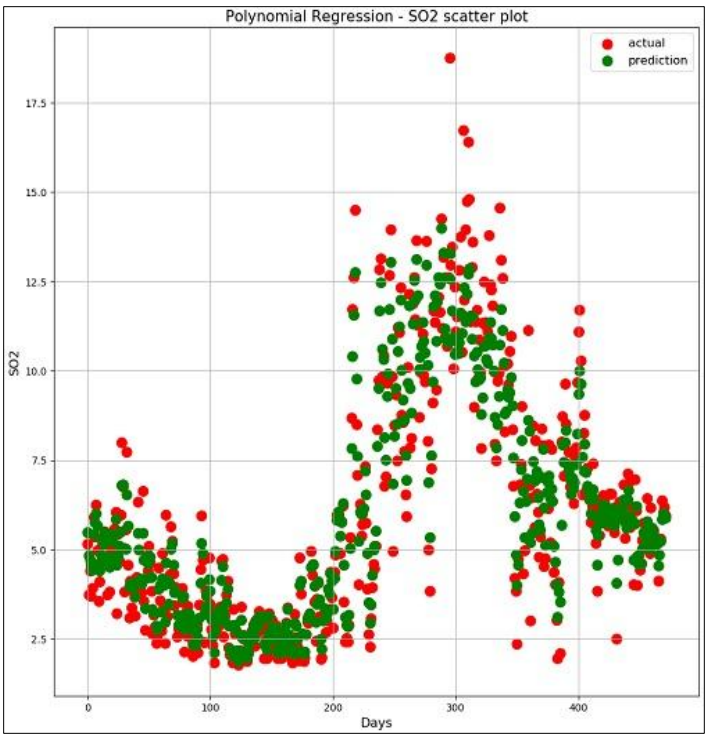
Polynomial Regression



PM10 Polynomial regression, FIG. 4.3

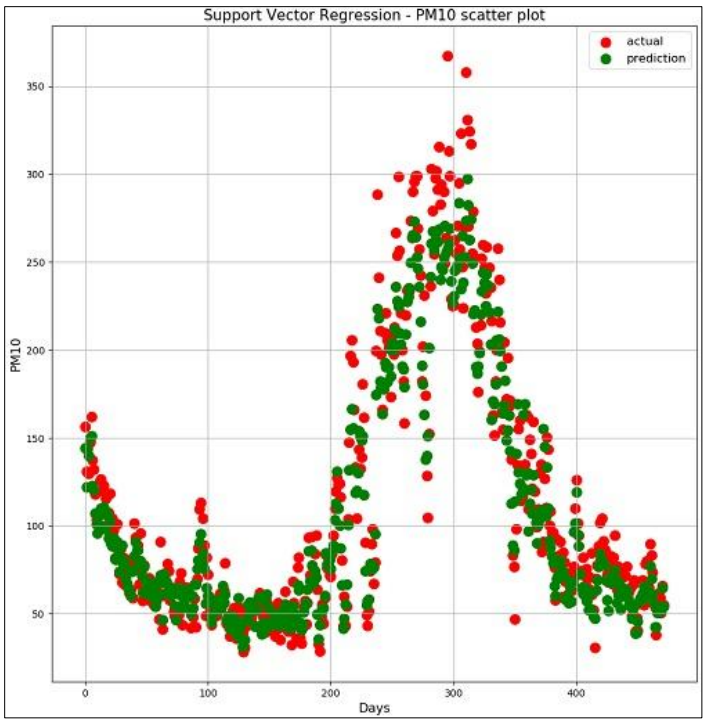


NO2 Polynomial regression, FIG. 4.4

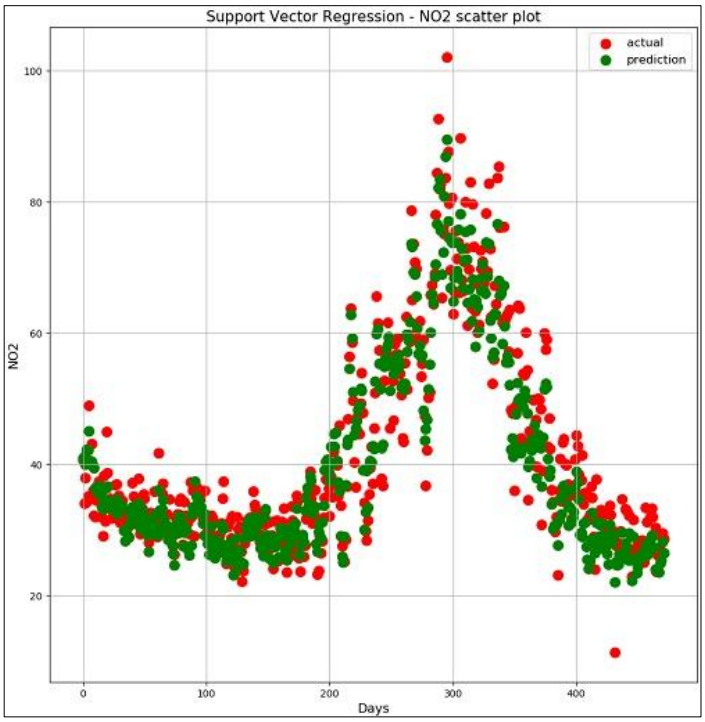


SO2 Polynomial regression, FIG. 4.5

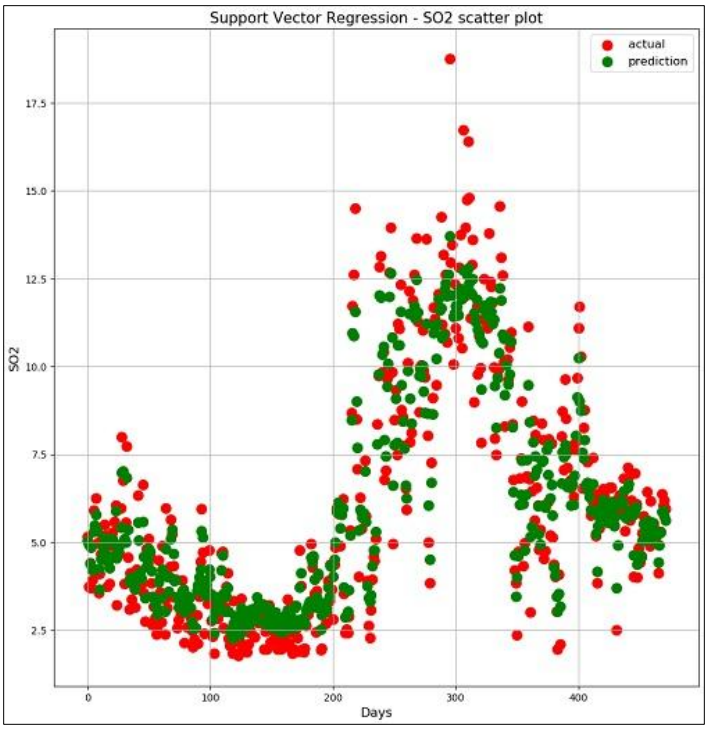
Support Vector Regression



PM10 Support vector regression, FIG. 4.6

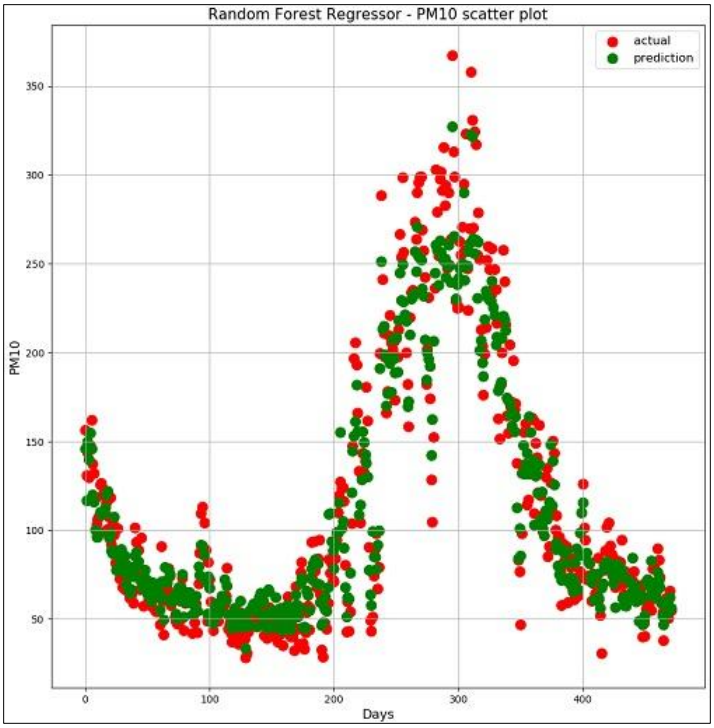


NO2 Support vector regression, FIG. 4.7

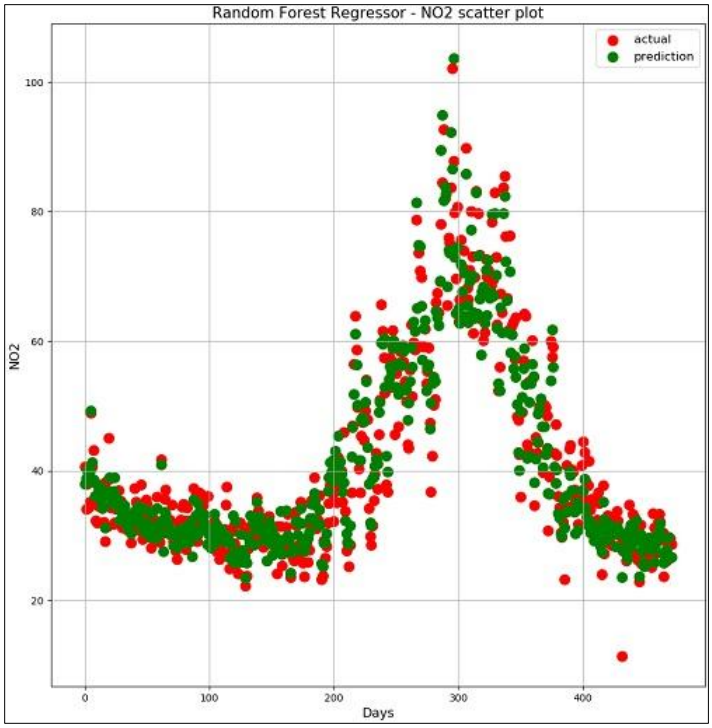


SO2 Support vector regression, FIG. 4.8

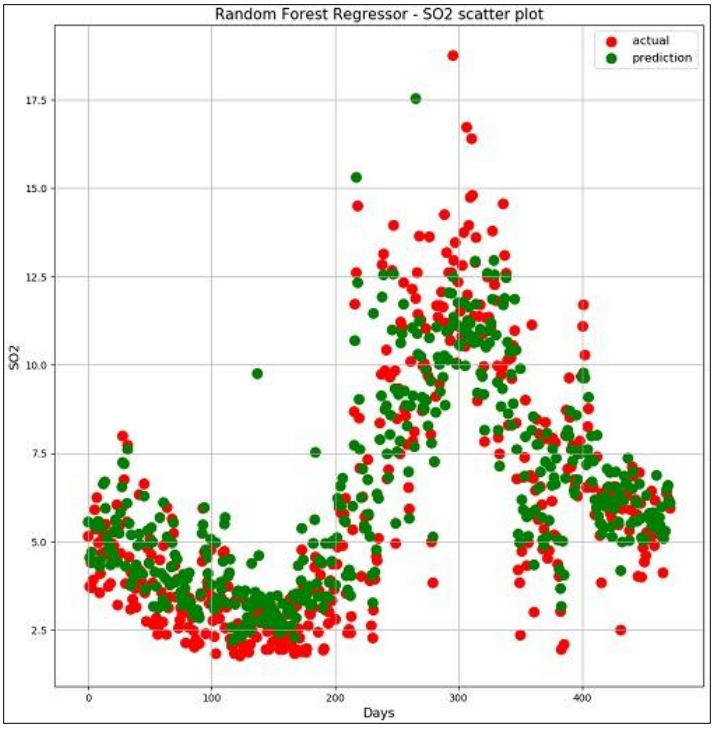
Random Forest



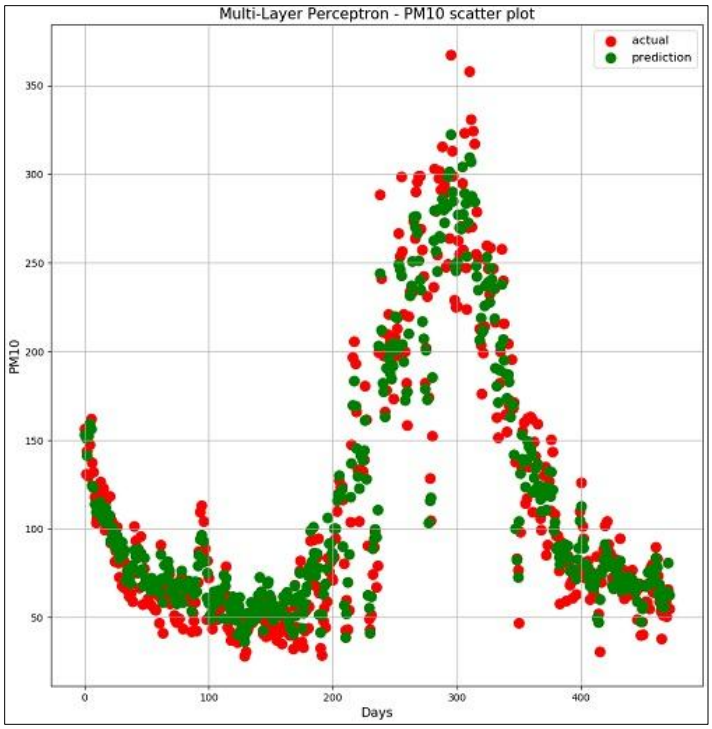
PM10 Random Forest, FIG. 4.9



NO2 Random Forest, FIG. 4.10

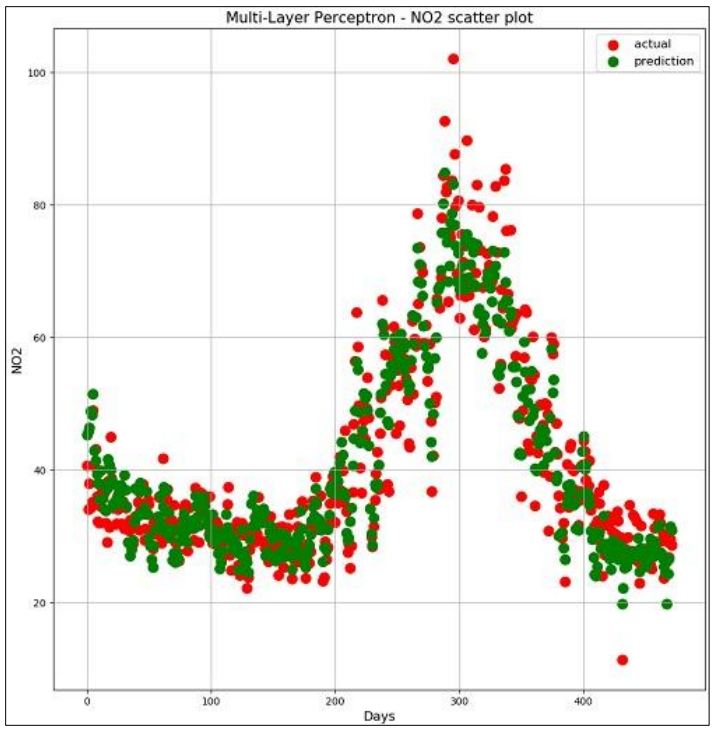


SO2 Random Forest, FIG. 4.11

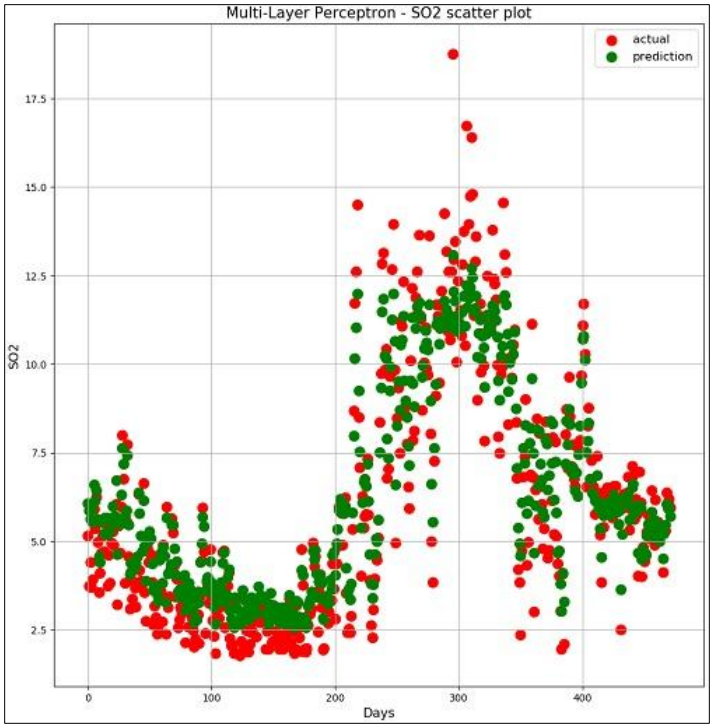


Multi-layer Perceptron Neural Network

PM10 Multilayer perceptron, FIG. 4.12

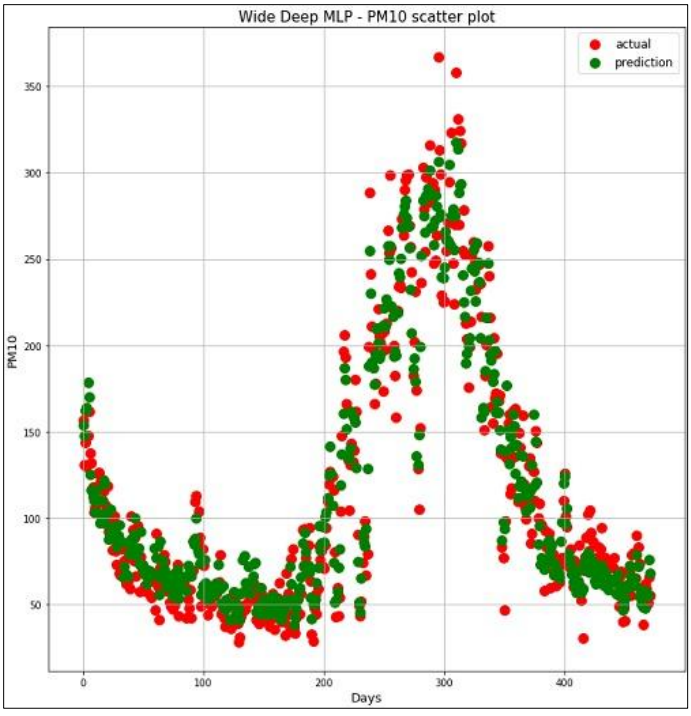


NO2 Multi layer Perceptron, FIG. 4.13

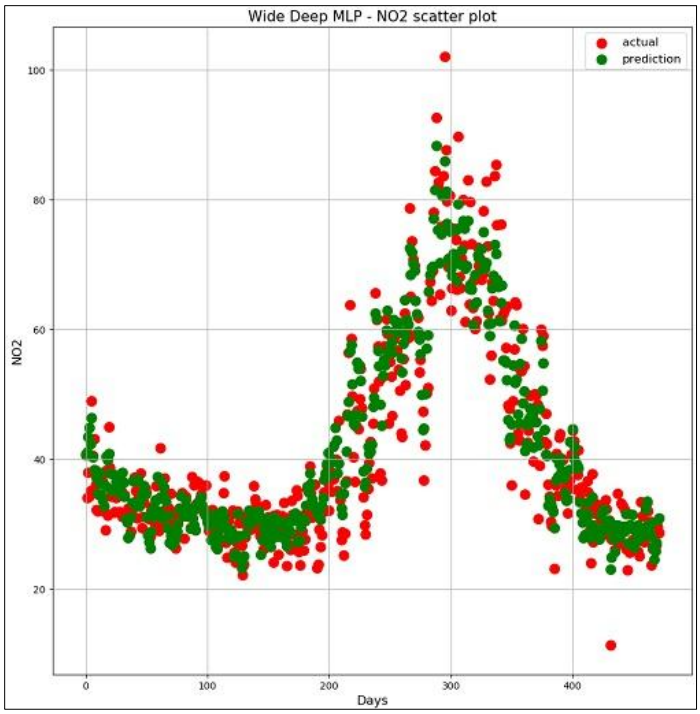


SO2 Multilayer Perceptron, FIG. 4.14

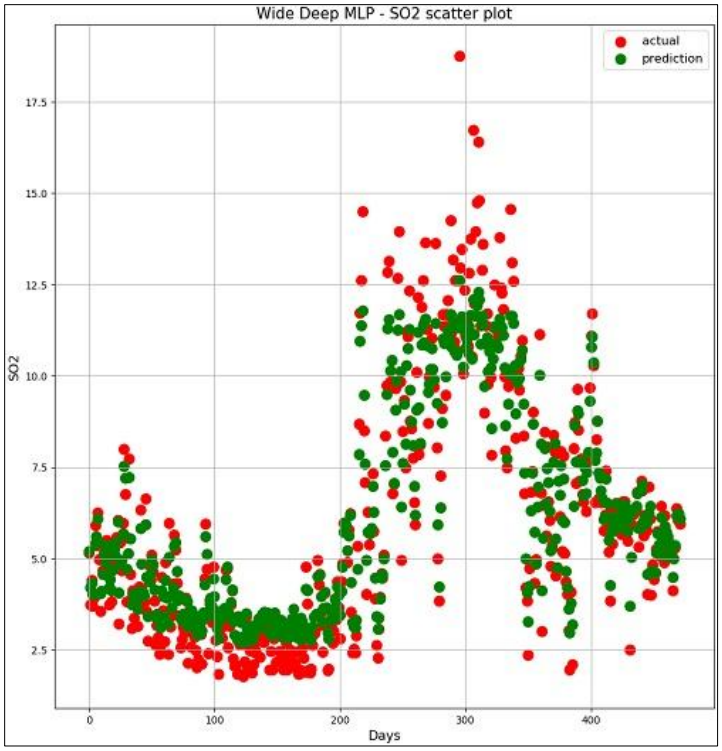
Wide Deep Neural Network



PM10 Wide deep neural network, FIG. 4.15

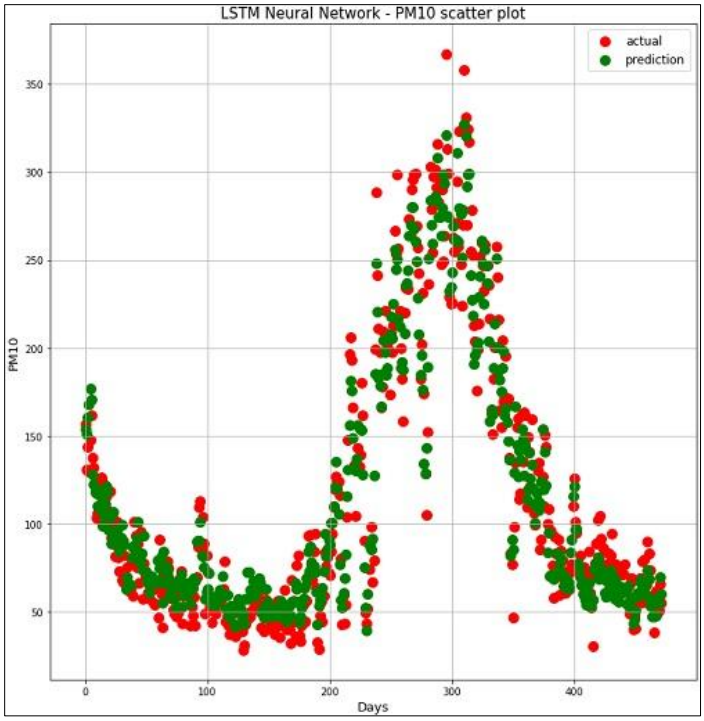


NO2 Wide deep neural network, FIG. 4.16

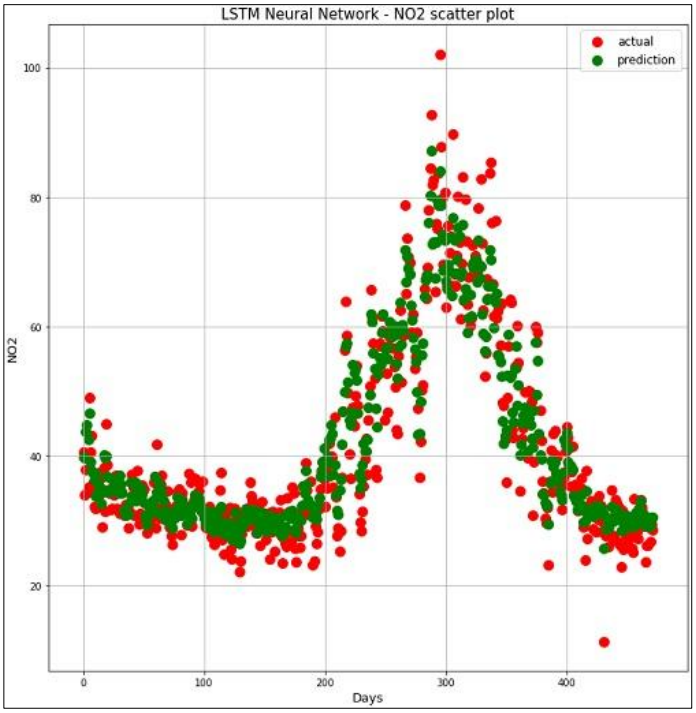


SO2 Wide deep neural network, FIG. 4.17

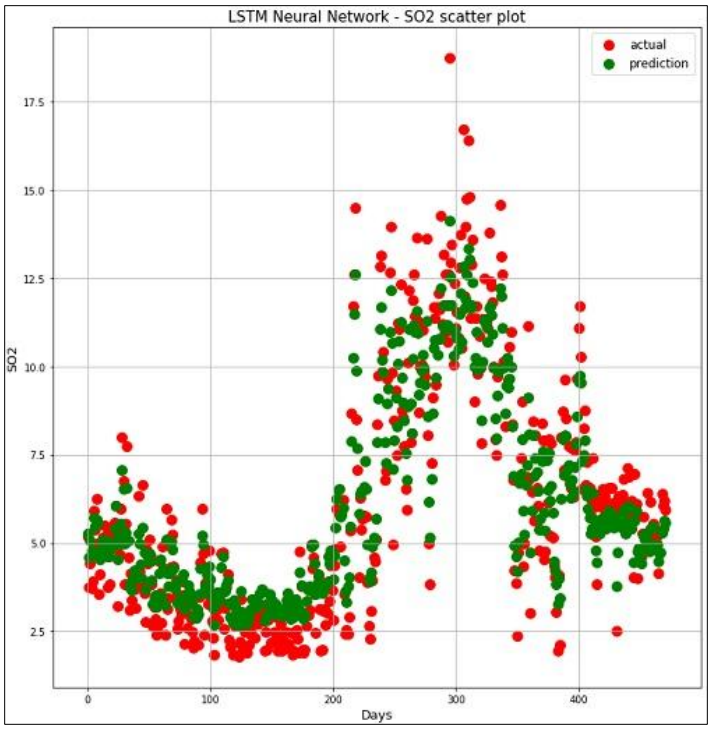
Long Short-Term Memory (LSTM) Recurrent Neural Network



PM10 LSTM neural network, FIG. 4.18

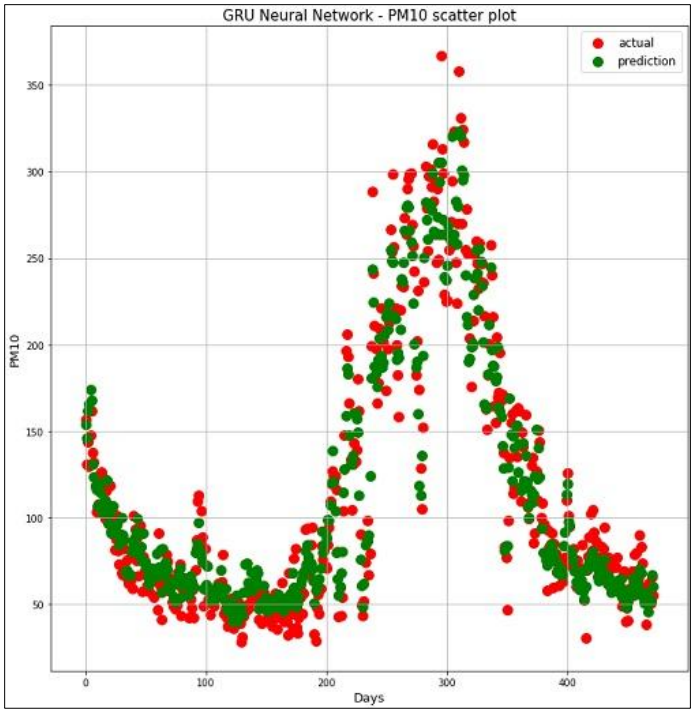


NO2 LSTM neural network, FIG. 4.19

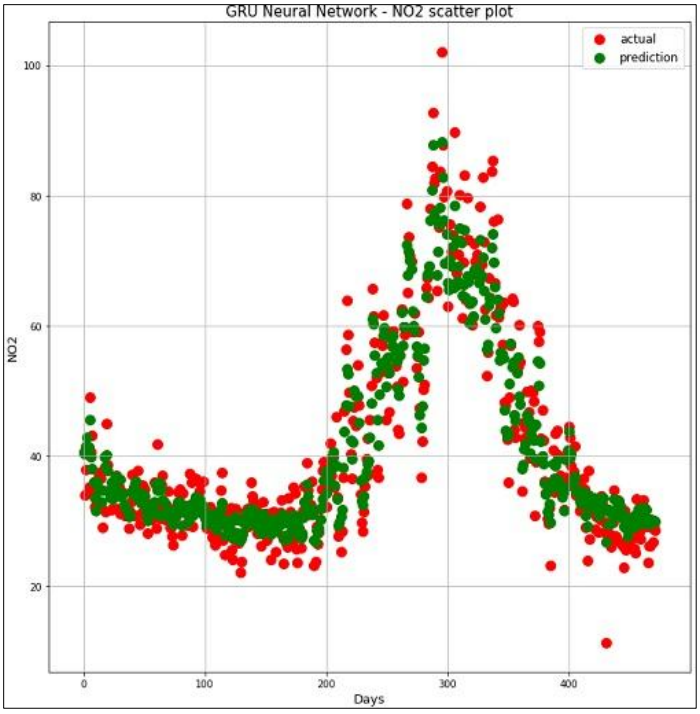


SO2 LSTM neural network, FIG. 4.20

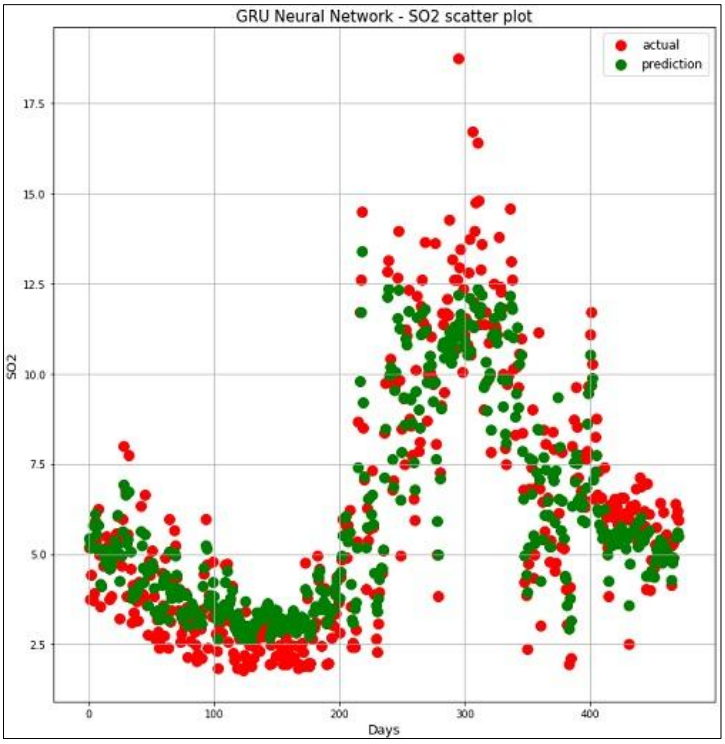
Gated Recurrent Unit (GRU)



PM10 GRU Neural network, FIG. 4.21



NO2 GRU Neural Network, FIG. 4.22



SO2 GRU Neural network, FIG. 4.23

Chapter V

Conclusion and Future scope

After the study and implementation of various machine learning algorithms, we have arrived at a solution with reasonable efficiency of prediction. PM_{10} and NO_2 were found to have a regular trend for a complete year and our models could effectively capture this trend as well as the change in their values that happen over longer periods of time. It can also be concluded that air quality parameters are very heavily dependent on past values as well as meteorological parameters. However, SO_2 values remain quite stable round the year and most models failed to produce good accuracy for its prediction. A comparative study of these algorithms also reveals that for our specific problem the performance of most well-known regression algorithms is quite on par each other.

For the modeling and prediction of $PM_{10,}$ 2^{nd} polynomial regression performed extremely well. Neural network models performed quite well and on par with polynomial regression. support vector regression, random forest regression had poorer performance and multi-linear regression was the worst case. This can be attributed to the fact that the data for PM_{10} shows a quadratic trend which cannot be modeled well by a linear algorithm. The other models which are non-linear thus performed better.

The data for NO_2 also shows a quadratic trend and hence the performance of multi-linear regression is also not as good as the other models. Random forest regression also gives results similar to multi-linear regression. Although poorer compared to other models, both the previously mentioned models still performed very well. Polynomial regression and support vector regression performed extremely well and on par with each other. Neural network models performed the best in this category and showed very promising accuracy. However, their gain in accuracy over polynomial and support vector regression is low especially considering the increase in complexity when neural networks are involved.

SO_2 values don't have any prominent trend and don't show great change over time. None of the models were capable of predicting SO_2 values with the kind of accuracy that were achieved for the other parameters. Random forest regression and multi-linear regression produced the poorest results. From our test set these models made a prediction error greater than 20% for nearly half of the predictions. Polynomial regression performed better but not by a great extent. Among the neural network models multi-layer perceptron performed well but its wide-deep variation was as poor as multi-linear regression. LSTM and GRU performed much better and had good accuracy.

However, by and far the best model was support vector regression which performed much better than all other models.

For our specific problem although neural network models performed consistently well their performance was not greatly superior compared to models like polynomial regression and support vector regression. In practical use cases an average of the predictions from the best models may be a better generalization of the trends and reduce error over long periods of usage. The models for PM_{10} and NO_2 have practically reasonable accuracy and can be used for real life prediction or for further study on this subject.

The effect of these air quality parameters on AQI values was studied and the results are quite unsurprising. PM_{10} has long been one of the greatest problems with air quality and our studies reveal that PM_{10} is almost always responsible for poor AQI. The values for PM_{10} reach strikingly high levels every year during the winter season and easily surpasses the effect of other pollutants on air quality. NO_2 is the next most influential pollutant and is very close to PM_{10} in its influence on AQI. NO_2 values also reach very high values during the winter season and is a major problem. SO_2 has much less influence on AQI and can be considered to be less of an immediate threat than the other two. As far as the how the data can change in the near future, we can make a few remarks. Firstly, PM_{10} seems to be peaking at nearly the same value every year and hence if present conditions do not worsen chances of its levels getting worse is small. Secondly, NO_2 shows a decrease in its peak value with every passing year and as such its impact on air quality can lessen and possibly it can get masked by the sustained impact of $PM_{10.}$ However, $PM_{2.5}$ which is another major pollutant that we could not study due to lack of data has been affecting the air quality greatly in the recent years. So, unless effective steps are taken by the authorities to reduce air pollution, every person in Kolkata will be exposed to hazardous levels of pollution every year.

APPENDIX

Code snippets for individual models :

Dataset Splitting

```
y = dataset[target_y][1:].values
X = dataset[parameters][:-1].values
train_length = round(X.shape[0] * (0.8))
X_train = np.array(X[:train_length])
y_train = np.array(y[:train_length])
X_test = np.array(X[train_length:])
y_test = np.array(y[train_length:])
```

Model Evaluation

```
from sklearn import metrics
y_error = predictions.ravel() - y_test.ravel()
y_error = np.array([abs(e) for e in y_error]).ravel()
for i in range(len(y_error)):
        if(y_error[i] < 0.20 * y_test[i]):
                count += 1
accuracy = count / len(predictions)
mean_abs_error = metrics.mean_absolute_error(y_test, predictions)
mean_sq_error = metrics.mean_squared_error(y_test, predictions)
root_mean_sq_error = np.sqrt(metrics.mean_squared_error(y_test, predictions))
score = metrics.r2_score(y_test, predictions)
```

Multivariable Linear Regression

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
predictions = lin_reg.predict(X_test)
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 2)
X_poly = poly_reg.fit_transform(X_train)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y_train)
X_transform = poly_reg.fit_transform(X_test)
predictions = lin_reg.predict(X_transform)
```

Support Vector Regression

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_fit = scaler_X.fit_transform(X_train)
y_fit = scaler_y.fit_transform(y_train.reshape(-1, 1))
regressor = SVR(kernel = 'rbf')
regressor.fit(X_fit, y_fit.ravel())
predictions = scaler_y.inverse_transform(regressor.predict(scaler_X.transform(X_test)))
predictions = predictions.ravel()
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
forest_regressor = RandomForestRegressor(n_estimators = 50)
forest_regressor.fit(X_train, y_train)
predictions = forest_regressor.predict(X_test)
```

Artificial Neural Networks

For the training of artificial neural networks, the data needs to be scaled using a scaler so that all values lie between 0 and 1. The code for performing this is as follows

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

After scaling the data, they can be used to train neural network models. The code snippets for the same are shown next,

Multi-Layer Perceptron

```
from sklearn.neural_network import MLPRegressor
from tensorflow import keras
#using sklearn
nn = MLPRegressor(hidden_layer_sizes=(75, 75, 75, ), activation='relu', max_iter= 200, solver='lbfgs',
n_iter_no_change=100, learning_rate_init=0.003, batch_size=28)
nn.fit(X_train, y_train)
predictions = nn.predict(X_test)
#using keras
model = keras.models.Sequential()
model.add(keras.layers.Dense(30, input_shape = X_train.shape[1:], activation = 'relu'))
model.add(keras.layers.Dense(30, activation = 'relu'))
model.add(keras.layers.Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
history = model.fit(X_train, y_train, validation_split = 0.1, epochs = 50, verbose = 0, batch_size = 28)
predictions = model.predict(X_test)
```

Wide-deep Multi-Layer Perceptron

```
X_a = dataset[wide_params][:-1]
X_b = dataset[deep_params][:-1]
y = dataset[target_param][1:]
X_train_a = np.array(X_a[:1900])
X_train_b = np.array(X_b[:1900])
X_test_a = np.array(X_a[1900:])
X_test_b = np.array(X_b[1900:])
y_train = np.array(y[:1900])
y_test = np.array(y[1900:])
a_scaler = MinMaxScaler()
b_scaler = MinMaxScaler()
X_train_a = a_scaler.fit_transform(X_train_a)
X_train_b = b_scaler.fit_transform(X_train_b)
X_test_a = a_scaler.transform(X_test_a)
X_test_b = b_scaler.transform(X_test_b)
input_layer_a = keras.layers.Input(shape = X_train_a.shape[1:])
input_layer_b = keras.layers.Input(shape = X_train_b.shape[1:])
hidden1 = keras.layers.Dense(30, activation = 'relu')(input_layer_a)
hidden2 = keras.layers.Dense(30, activation = 'relu')(hidden1)
hidden3 = keras.layers.Dense(30, activation = 'relu')(hidden2)
concat = keras.layers.concatenate([input_layer_b, hidden3])
output_layer = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs = [input_layer_a, input_layer_b], outputs = [output_layer])
model.compile(optimizer = keras.optimizers.Adam(0.001), loss = 'mse')
history = model.fit((X_train_a, X_train_b),y_train, epochs= 200, validation_split = 0.1,batch_size =28)
predictions = model.predict((X_test_a, X_test_b))
```

```
from tensorflow import keras
model = keras.models.Sequential()
model.add(keras.layers.LSTM(30, input_shape = (1, X_train.shape[1:][1]), activation = 'relu',
return_sequences = True))
model.add(keras.layers.LSTM(30, activation = 'relu', return_sequences = True))
model.add(keras.layers.LSTM(30, activation = 'relu', return_sequences = True))
model.add(keras.layers.Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = keras.optimizers.Adam(0.001))
history = model.fit(X_train, y_train, validation_split = 0.1, epochs = 200, batch_size = 28)
predictions = model.predict(X_test)
```

Long Short-Term Memory (LSTM) Recurrent Neural Network

Gated Recurrent Unit (GRU)

```
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
model = keras.models.Sequential()
model.add(keras.layers.GRU(30, input_shape = (1, X_train.shape[1:][1]), activation = 'relu',
recurrent_activation = 'relu', return_sequences = True))
model.add(keras.layers.GRU(30, activation = 'relu', recurrent_activation = 'relu', return_sequences =
True))
model.add(keras.layers.GRU(30, activation = 'relu', recurrent_activation = 'relu', return_sequences =
True))
model.add(keras.layers.Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = keras.optimizers.Adam(0.001))
history = model.fit(X_train, y_train, validation_split = 0.1, epochs = 50, verbose = 0, batch_size = 28)
predictions = model.predict(X_test)
```

References

1. Kanchan, A. K. Gorai, and P. Goyal, "A review on air quality indexing system," *Asian Journal of Atmospheric Environment*, vol. 9, no. 2, pp. 101–113, 2015.View at: Publisher Site | Google Scholar
2. M. A. Elangasinghe, N. Singhal, K. N. Dirks, and J. A. Salmond, "Development of an ANN–based air pollution forecasting system with explicit knowledge through sensitivity analysis," *Atmospheric Pollution Research*, vol. 5, no. 4, pp. 696–708, 2014.View at: Publisher Site | Google Scholar
3. Air Quality Prediction by Machine Learning Methods by Huiping Peng.
4. A systematic review of data mining and machine learning for air pollution epidemiology by Colin Bellinger,Mohomed Shazan Mohomed Jabbar,Osmar Zaïane and Alvaro Osornio-Vargas
5. HISYCOL a hybrid computational intelligence system for combined machine learning: the case of air pollution modeling in Athens Ilias Bougoudis1, Konstantinos Demertzis1,Lazaros Iliadis1.
6. E. Y. Bezuglaya, A. B. Shchutskaya, and I. V. Smirnova, "Air pollution index and interpretation of measurements of toxic pollutant concentrations," *Atmospheric Environment Part A, General Topics*, vol. 27, no. 5, pp. 773–779, 1993.View at: Publisher Site | Google Scholar
7. G. Kyrkilis, A. Chaloulakou, and P. A. Kassomenos, "Development of an aggregate Air Quality Index for an urban Mediterranean agglomeration: Relation to potential health effects," *Environment International*, vol. 33, no. 5, pp. 670–676, 2007.View at: Publisher Site | Google Scholar
8. https://data.gov.in/catalog/historical-daily-ambient-air-quality-data.
9. B. Cyganek, B. Krawczyk, and M. Woźniak, "Multidimensional data classification with chordal distance based kernel and Support Vector Machines," *Engineering Applications of Artificial Intelligence*, vol. 46, pp. 10–22, 2015.View at: Publisher Site | Google Scholar
10. "The official website of Pollution Control Board of India," http://cpcb.nic.in/.View at: Google Scholar
11. http://www.cpcb.nic.in/EIS_on_GIS.pdf.