Project Report

on

“Efficiency of different ML/DL models in predicting Air Quality”

Submitted in Fulfillment of the Course

Study Oriented Project – CS F266

Under the guidance and supervision of

Dr L. Rajya Lakshmi

Faculty, Department of Computer Science & Information Systems

Birla Institute of Technology & Science, Pilani, 333031

Rajasthan, INDIA

Acknowledgements

I would like to express my deep and sincere gratitude for the guidance provided by my supervisor, Dr. L Rajya Lakshmi, under whose guidance this project became a success. I came across several new topics and learnt many things through this experience and am very thankful for this opportunity provided by her. Her guidance, professionalism, support and motivation helped me see this project through, and I would like to express my immense gratitude for her help.

Sincerely,

Lakshay Munjal

2019A7PS0047P

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI (RAJASTHAN)

Abstract

Owing to global industrialization and urbanisation, air pollution has been one of the most serious environmental problems in the twenty-first century. Accurate air quality forecasts are needed for its mitigation. Deep learning has recently emerged as a general-purpose technology for extracting complex information from vast amounts of data and very large networks of neurons, and hence has the ability to push air quality forecasting to new heights. We provide a quick rundown of recent attempts to use deep learning approaches in air quality forecasting. First we define various methods and architectures of deep networks and their relevance to explore the non-linear spatio-temporal features across multiple scales of air pollution. We then compare these with various supervised machine learning models to see how they hold up against traditional regression approaches. We finally give a conclusion and discuss the viability of these networks in Air Quality Forecasting.

TABLE OF CONTENTS

1. Introduction

2. Related Works

3. Dataset

4. Various Models

5. Model definitions and Observations

6. Conclusions

1. Introduction

Air Pollution is a major environmental problem that is gaining worldwide notice and is a major issue in many developing countries. Obtaining real-time air quality data is critical for air pollution management and shielding us from the negative health effects of pollution. As a result, air quality estimation is needed to better represent the evolving pattern of air emissions, provide timely and full environmental quality information for environmental management decisions, and prevent severe air pollution incidents.

Recently, deep learning, a new potential machine learning methodology, has attracted considerable academic and industrial attention and has been successfully applied to image classification, natural language processing, prediction task, object detection, artificial intelligence, motion modeling, etc.

Deep learning algorithms use multilayered frameworks to extricate the inherent properties of the data layer by layer from the lowest to the highest level and to identify emblematic structures in the data. The innate complexity of the air quality process causes its spatial distribution and over-the-time trends to be influenced by numerous factors , air pollutant emissions and deposits, weather conditions, traffic flow, human activities being a few of many. This situation has created complications in using conventional flat models, when used particularly to provide a decent rendition of air quality characteristics. These algorithms can extract representative air quality features without prior knowledge, leading to a more efficient interpretation of air quality predictions.

2. Literature Survey

Multiple linear regression (MLR) (Li et al. 2011) model and the auto regression moving average (ARMA) model are commonly used for air quality prediction. However, these methods usually yield limited accuracy due to their inability to model nonlinear patterns. A promising alternative is ANNs like SVR (Nieto et al. 2013).

Deep networks' architectures assess their ability to derive complex nonlinear features from high-dimensional datasets across scales. Deep artificial neural networks are distinguished from shallow artificial neural networks by the presence of several layers of neurons. Qi Liao et al. 2020 uses Simple RNN, LSTM and a GRU network to build the temporal portion of the spatiotemporal model.

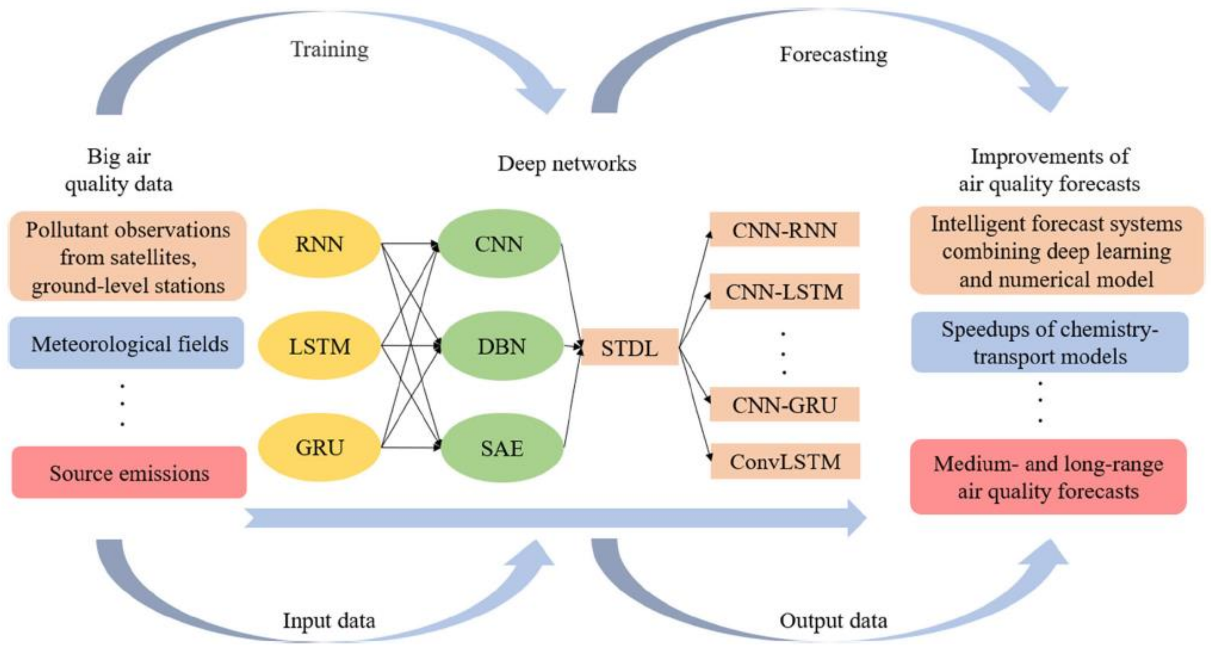

Training
Forecasting
Big air quality data
Deep networks
Improvements of air quality forecasts
Pollutant observations from satellites, ground-level stations
Meteorological fields
Source emissions
RNN
LSTM
GRU
CNN
DBN
SAE
STDL
CNN-RNN
CNN-LSTM
CNN-GRU
ConvLSTM
Intelligent forecast systems combining deep learning and numerical model
Speedups of chemistry-transport models
Medium- and long-range air quality forecasts
Input data
Output data

Figure 1 Spatio Temporal Model

Since the focus of this paper is on the temporal efficiency of these models we have only used the temporal module of this spatiotemporal model.

Xiang Li et al. 2016 uses an LR paired with a 4 layer stacked autoencoder to create a spatiotemporal model.

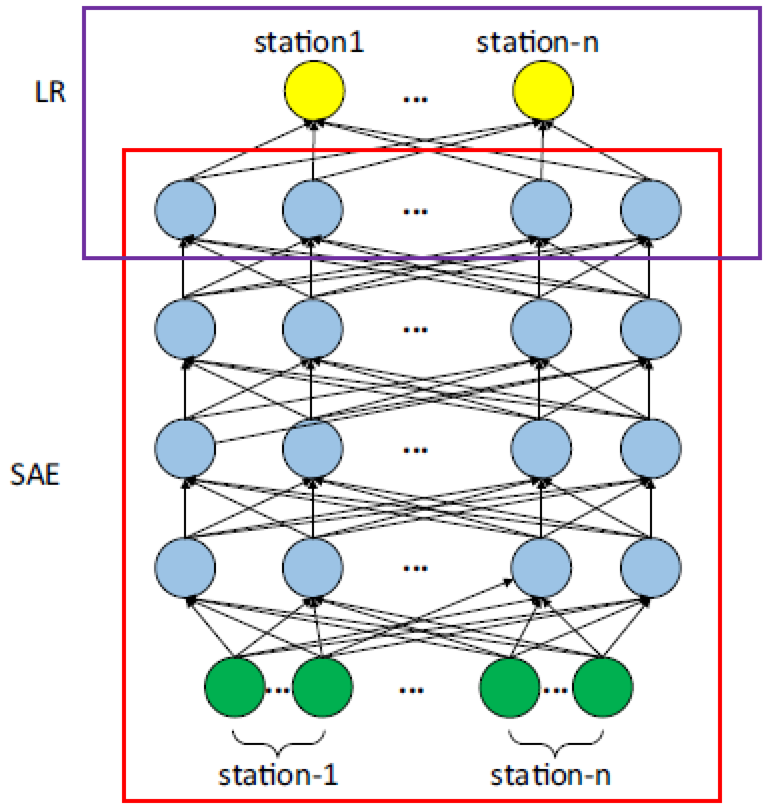
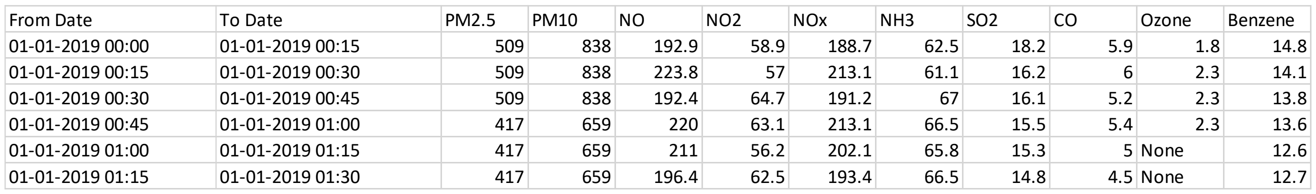


Figure 2 Stacked AutoEncoder

A SAE is a concatenation of autoencoders which the outputs of the autoencoder stacked on the layer below are wired to the inputs of the successive layer. More specifically, for a SAE with L layers, the first layer is trained using the training set as the input. After obtaining the first hidden layer, the output of the kth ($k < L$) hidden layer is utilized as the input for the ($k + 1$) hidden layer. Using this method, sequential autoencoders can be stacked hierarchically. Each hidden layer is a higher-level abstraction of the previous layer, and the last hidden layer contains high-level structure and representative information of the input, which are more effective for the successive prediction

3. Dataset

The concentration of **PM2.5, PM10, NO, NO2, NOx, NH3, SO2, CO, Ozone** and **Benzene** for the city of New Delhi was downloaded from the official website of the Central Pollution Control Board. The Dataset is specifically from the station at **Dwarka-Sector 8, Delhi**, under the DPCC. The data is recorded at every **15 min interval** from **1st January 2019** to **1st January 2021**.

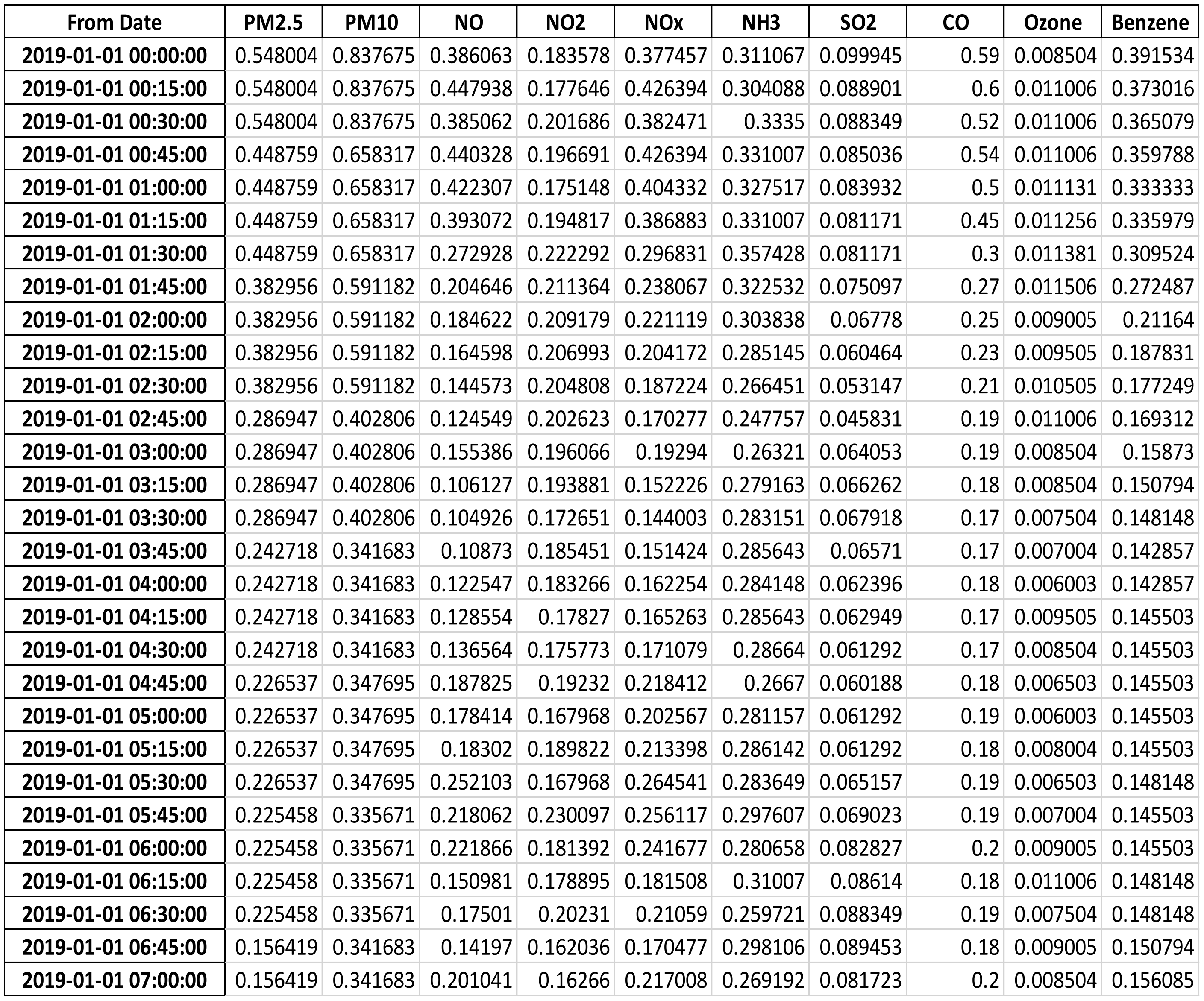


From Date	To Date	PM2.5	PM10	NO	NO2	NOx	NH3	SO2	CO	Ozone	Benzene
01-01-2019 00:00	01-01-2019 00:15	509	838	192.9	58.9	188.7	62.5	18.2	5.9	1.8	14.8
01-01-2019 00:15	01-01-2019 00:30	509	838	223.8	57	213.1	61.1	16.2	6	2.3	14.1
01-01-2019 00:30	01-01-2019 00:45	509	838	192.4	64.7	191.2	67	16.1	5.2	2.3	13.8
01-01-2019 00:45	01-01-2019 01:00	417	659	220	63.1	213.1	66.5	15.5	5.4	2.3	13.6
01-01-2019 01:00	01-01-2019 01:15	417	659	211	56.2	202.1	65.8	15.3	5	None	12.6
01-01-2019 01:15	01-01-2019 01:30	417	659	196.4	62.5	193.4	66.5	14.8	4.5	None	12.7

Pre-Processing of Dataset

1. In the end, we had 70176 lines of data.
2. The gaps in data, represented by 'None' are filled by linear interpolation so as the preserve the linear dependencies of the data on time of data.
3. The data is normalized using MinMax Normalization. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1.
4. We also split the dataset into testing and training set in the ratio 1:3. The testing set is taken from the end of the dataset so as to preserve the linear dependency of the data.

Finally our dataset looks like this:



From Date	PM2.5	PM10	NO	NO2	NOx	NH3	SO2	CO	Ozone	Benzene
2019-01-01 00:00:00	0.548004	0.837675	0.386063	0.183578	0.377457	0.311067	0.099945	0.59	0.008504	0.391534
2019-01-01 00:15:00	0.548004	0.837675	0.447938	0.177646	0.426394	0.304088	0.088901	0.6	0.011006	0.373016
2019-01-01 00:30:00	0.548004	0.837675	0.385062	0.201686	0.382471	0.3335	0.088349	0.52	0.011006	0.365079
2019-01-01 00:45:00	0.448759	0.658317	0.440328	0.196691	0.426394	0.331007	0.085036	0.54	0.011006	0.359788
2019-01-01 01:00:00	0.448759	0.658317	0.422307	0.175148	0.404332	0.327517	0.083932	0.5	0.011131	0.333333
2019-01-01 01:15:00	0.448759	0.658317	0.393072	0.194817	0.386883	0.331007	0.081171	0.45	0.011256	0.335979
2019-01-01 01:30:00	0.448759	0.658317	0.272928	0.222292	0.296831	0.357428	0.081171	0.3	0.011381	0.309524
2019-01-01 01:45:00	0.382956	0.591182	0.204646	0.211364	0.238067	0.322532	0.075097	0.27	0.011506	0.272487
2019-01-01 02:00:00	0.382956	0.591182	0.184622	0.209179	0.221119	0.303838	0.06778	0.25	0.009005	0.21164
2019-01-01 02:15:00	0.382956	0.591182	0.164598	0.206993	0.204172	0.285145	0.060464	0.23	0.009505	0.187831
2019-01-01 02:30:00	0.382956	0.591182	0.144573	0.204808	0.187224	0.266451	0.053147	0.21	0.010505	0.177249
2019-01-01 02:45:00	0.286947	0.402806	0.124549	0.202623	0.170277	0.247757	0.045831	0.19	0.011006	0.169312
2019-01-01 03:00:00	0.286947	0.402806	0.155386	0.196066	0.19294	0.26321	0.064053	0.19	0.008504	0.15873
2019-01-01 03:15:00	0.286947	0.402806	0.106127	0.193881	0.152226	0.279163	0.066262	0.18	0.008504	0.150794
2019-01-01 03:30:00	0.286947	0.402806	0.104926	0.172651	0.144003	0.283151	0.067918	0.17	0.007504	0.148148
2019-01-01 03:45:00	0.242718	0.341683	0.10873	0.185451	0.151424	0.285643	0.06571	0.17	0.007004	0.142857
2019-01-01 04:00:00	0.242718	0.341683	0.122547	0.183266	0.162254	0.284148	0.062396	0.18	0.006003	0.142857
2019-01-01 04:15:00	0.242718	0.341683	0.128554	0.17827	0.165263	0.285643	0.062949	0.17	0.009505	0.145503
2019-01-01 04:30:00	0.242718	0.341683	0.136564	0.175773	0.171079	0.28664	0.061292	0.17	0.008504	0.145503
2019-01-01 04:45:00	0.226537	0.347695	0.187825	0.19232	0.218412	0.2667	0.060188	0.18	0.006503	0.145503
2019-01-01 05:00:00	0.226537	0.347695	0.178414	0.167968	0.202567	0.281157	0.061292	0.19	0.006003	0.145503
2019-01-01 05:15:00	0.226537	0.347695	0.18302	0.189822	0.213398	0.286142	0.061292	0.18	0.008004	0.145503
2019-01-01 05:30:00	0.226537	0.347695	0.252103	0.167968	0.264541	0.283649	0.065157	0.19	0.006503	0.148148
2019-01-01 05:45:00	0.225458	0.335671	0.218062	0.230097	0.256117	0.297607	0.069023	0.19	0.007004	0.145503
2019-01-01 06:00:00	0.225458	0.335671	0.221866	0.181392	0.241677	0.280658	0.082827	0.2	0.009005	0.145503
2019-01-01 06:15:00	0.225458	0.335671	0.150981	0.178895	0.181508	0.31007	0.08614	0.18	0.011006	0.148148
2019-01-01 06:30:00	0.225458	0.335671	0.17501	0.20231	0.21059	0.259721	0.088349	0.19	0.007504	0.148148
2019-01-01 06:45:00	0.156419	0.341683	0.14197	0.162036	0.170477	0.298106	0.089453	0.18	0.009005	0.150794
2019-01-01 07:00:00	0.156419	0.341683	0.201041	0.16266	0.217008	0.269192	0.081723	0.2	0.008504	0.156085

4. Various Models

4.1. K-Nearest Neighbor

This method assign a predicted value to a new observation based on the plurality or mean (sometimes weighted) of its k “Nearest Neighbors” in the training set. Given an infinite amount of data, any observation will have many “neighbors” that are arbitrarily near with respect to all measured characteristics, and the variability of their outcomes will provide as precise a prediction as is theoretically possible barring a perfectly and completely specified model. However, given that we never have an infinite amount of data, the actual utility of this asymptotic property is questionable, especially for modest datasets.

The algorithm predicts the values of new data points based on ‘feature similarities'. This implies that a value is given to the new point depending on how similar it is to the points in the training set. The distance between the new point and the training points is determined, and the K data points that are closest to the new point are chosen. The new point's projection is the average of these K data points. This is an instance-based learning (or lazy learning) technique in which the function is only addressed locally and all estimates are delayed before classification or regression.

4.2. SVR

Support Vector Regression uses the same basic idea as Support Vector Machine (SVM), a popularly and widely used algorithm for classification problems, but applies it to predict real values rather than a class. SVR acknowledges the presence of non-linearity in the data and provides a proficient prediction model. SVR allows us to determine how much error is reasonable in our model and will fit the data with an appropriate line (or hyperplane in higher dimensions). The algorithm's goal is to minimise the coefficients, specifically the l2-norm of the coefficient vector, rather than the squared error. We treat the error term in the constraints, where we set the absolute error to be less than or equal to a given margin, called the maximum error (epsilon). We may adjust epsilon to achieve the model's desired accuracy.

In conclusion, SVR is a powerful algorithm that allows us to choose how tolerant we are of errors, both in terms of an acceptable error margin(epsilon) and in terms of tuning our tolerance for exceeding that acceptable error rate and acknowledges the presence of non-linearity in the data and provides a proficient prediction model.

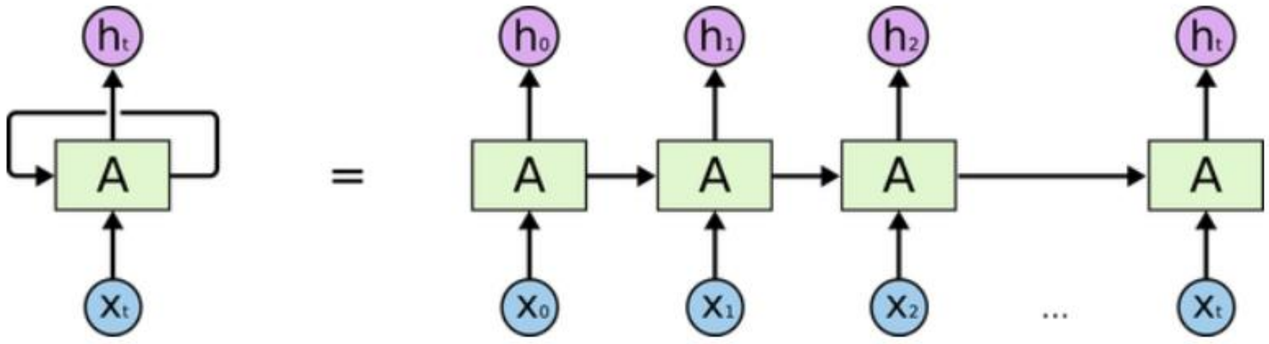
4.3. Decision tree

Decision trees construct regression or classification models based on tree structures. It incrementally breaks down a dataset into smaller and smaller subsets while also developing an associated decision tree. The end product is a tree consisting of decision nodes and leaf nodes. Each branch of a decision node represents a value for the attribute being evaluated. A decision on the numerical goal is represented by a leaf node. The root node is the topmost decision node in a tree that corresponds to the best predictor. The mean squared error is commonly used in decision trees regression to determine if a node should be split into two or more sub-nodes.
The resulting are three types of nodes in this tree-structured classifier. The Root Node is the first node in the graph, and it represents the entire sample. It can be further divided into nodes. The features of a data set are represented

by the interior nodes, and the decision rules are represented by the branches. Finally, the result is represented by the Leaf Nodes. This algorithm is extremely useful for resolving decision-making issues in the following way- A specific data point is traversed through the entire tree by answering True/False questions until it reaches the leaf node. The average of the dependent variable's value in that particular leaf node is the final estimate. The Tree is able to predict a proper value for the data point after several iterations..

4.4. Recurrent Neural Networks

RNNs are variants of feedforward neural networks but have self-connection of neurons cyclic structure into the network. Thus, input data can be memorized, and sequences of data can influence network outputs through self-connected neurons. A Recurrent Neural Network's main advantage is that it has at least one feed-back link, allowing activations to circle back on themselves. This allows the networks to do temporal processing and learn sequences, such as sequence recognition/reproduction or temporal association/prediction, for example. Recurrent neural network architectures can have many different forms. One common type consists of a standard Multi-Layer Perceptron (MLP) plus added loops. These can exploit the powerful non-linear mapping capabilities of the MLP, and also have some form of memory. Others have more uniform architectures, with every neuron theoretically attached to every other neuron, and stochastic activation mechanisms. Learning can be accomplished using similar gradient descent procedures to those that contribute to the back-propagation algorithm with feed-forward networks for simple architectures and deterministic activation functions.



Taking advantage of their memory characteristics, RNNs outperform FNNs in many applications. However, RNNs may fail to capture long time dependencies in input data, and it may face the problems of vanishing and exploding gradients when the time of training is too long.

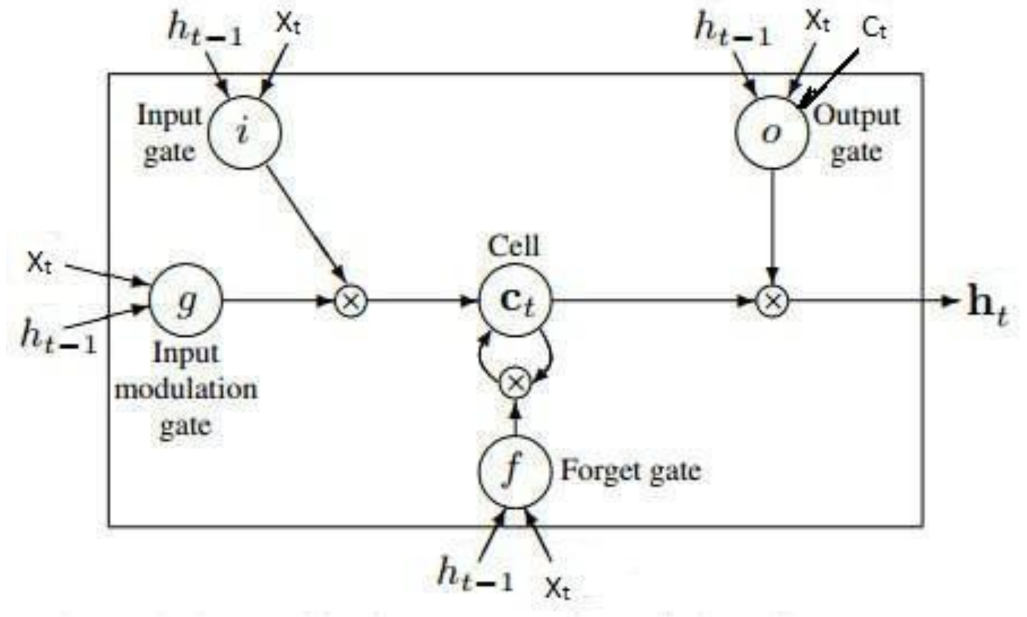
4.5. Long Short-Term Memory Networks

LSTMs are a particular kind of RNN that can learn long-term dependencies. They are still commonly used and perform exceptionally well on a wide range of issues. LSTMs was specifically designed to prevent the issue of long-term dependence. It is basically their default behavior to remember facts over long stretches of time. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

During the gradient back-propagation process of any typical recurrent neural network, the gradient signal can be multiplied by the weight matrix associated with the relations between the neurons of the recurrent hidden layer a significant number of times (as many as the number of time steps). This means that the magnitude of weights in the transition matrix can have a strong impact on the learning process.

If the weights in this matrix are small (or, more formally, if the weight matrix's leading eigenvalue is less than 1.0), vanishing gradients will occur, in which the gradient signal becomes so small that learning becomes very sluggish or stops operating entirely. It may also make understanding long-term dependencies in the data more complex. In contrast, if the weights in this matrix are high (or, to put it another way, if the leading eigenvalue of the weight matrix is greater than 1.0), the gradient signal will become so large that learning diverges. Exploding gradients is a term used to describe this phenomenon.

To solve the Vanishing Gradient Problem, we use memory blocks. The memory blocks are made up of three different kinds of nonlinear multiplicative gates: input, output, and forget. The multiplicative gates regulate memory block processing and decide whether or not the input data should be recalled. The input gate controls the flow of cell activation from input into a memory cell, while output gate controls the flow of output from a memory cell into other nodes.



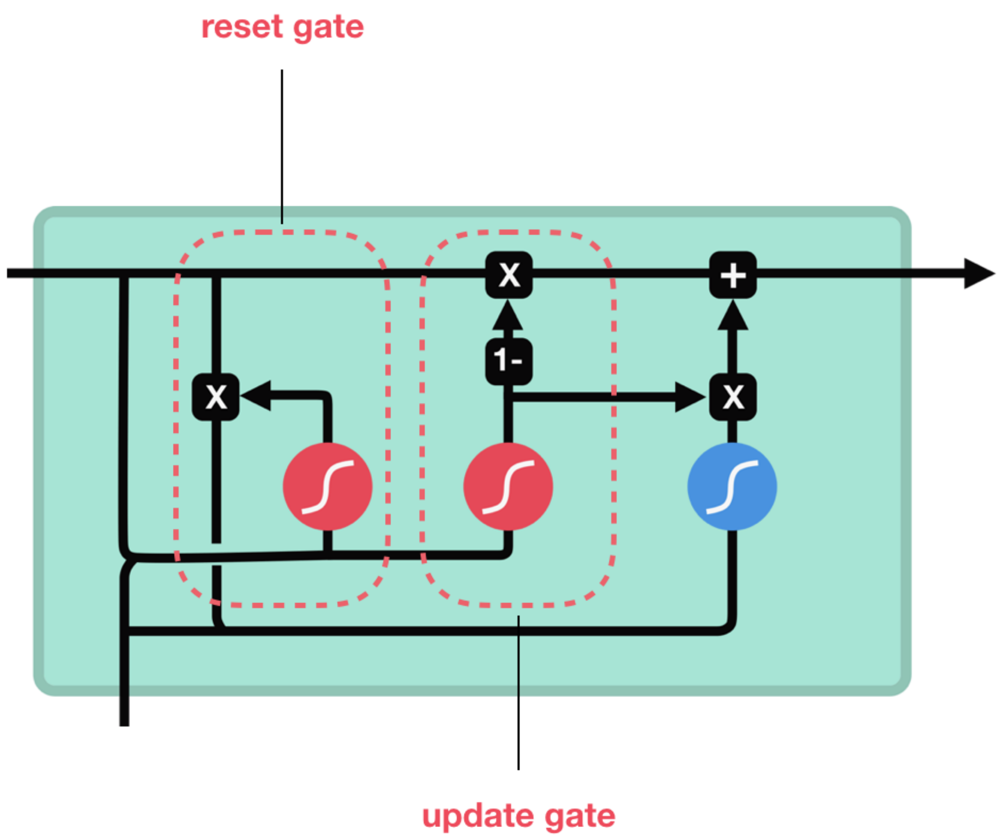
4.6. Gated Recurrent Unit networks

GRUs are simplified versions of the LSTM networks. It has only three gates, unlike LSTM, and does not keep track of the Internal Cell State. The information contained in an LSTM recurrent unit's Internal Cell State is inserted into the Gated Recurrent Unit's secret state. This aggregated data is forwarded to the next Gated Recurrent Unit. The different gates of a GRU are as described below:-

1. **Update Gate:** It decides how much historical experience must be passed on to future generations. It is analogous to the Output Gate in an LSTM recurrent unit.
2. **Reset Gate:** It decides how much of one's previous experience can be forgotten. It's similar to how an LSTM recurrent device combines the Input Gate and the Forget Gate.
3. **Current Memory Gate**: During a standard conversation on Gated Recurrent Unit Network, it is often ignored. It is a sub-component of the Reset Gate, much as the Input Modulation Gate is a sub-component of the Input Gate, and it is used to inject non-linearity into the input as well as making the input Zero-mean. Another justification for making it a sub-component of the Reset gate is to reduce the impact of past data on new data being transferred into the future.

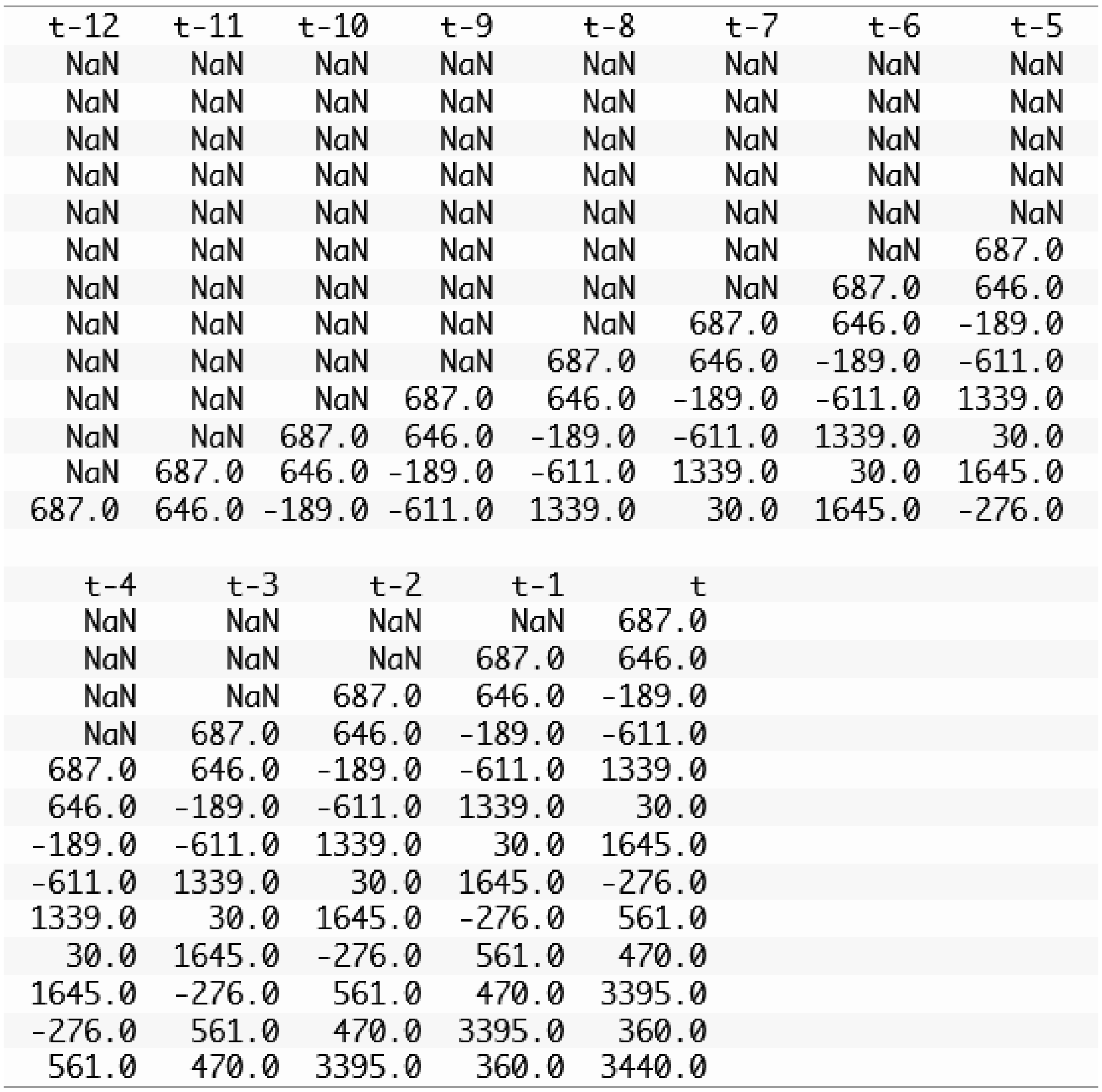
As compared to a general Recurrent Neural Network, the basic work-flow of a Gated Recurrent Unit Network is identical. The only distinction between the two is in the internal functioning of each recurrent unit, since Gated Recurrent Unit networks are made up of gates that modulate the current input and the previous secret state.

When compared to LSTM, the benefit of using GRU is that it has less parameters and thus requires less computing power to train. GRUs are a little faster to practise than LSTMs because they have less tensor operations. There is no straightforward winner when it comes to which is best. Researchers and engineers usually test them to see which one is best for their use.



5. Model definitions and Observations

To preserve the temporal relationships in the ML models, the data from t-16 to t-1 units is included as the features of the t^th unit. This way the context formed by the recent history of an input can be used to more accurately predict its future. Data appears of the form:



```
  t-12   t-11   t-10    t-9     t-8     t-7     t-6     t-5
   NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
   NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
   NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
   NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
   NaN    NaN    NaN    NaN     NaN     NaN     NaN     NaN
   NaN    NaN    NaN    NaN     NaN     NaN     NaN   687.0
   NaN    NaN    NaN    NaN     NaN     NaN   687.0   646.0
   NaN    NaN    NaN    NaN     NaN   687.0   646.0  -189.0
   NaN    NaN    NaN    NaN   687.0   646.0  -189.0  -611.0
   NaN    NaN    NaN  687.0   646.0  -189.0  -611.0  1339.0
   NaN    NaN  687.0  646.0  -189.0  -611.0  1339.0    30.0
   NaN  687.0  646.0 -189.0  -611.0  1339.0    30.0  1645.0
 687.0  646.0 -189.0 -611.0  1339.0    30.0  1645.0  -276.0

    t-4     t-3     t-2     t-1       t
    NaN     NaN     NaN     NaN   687.0
    NaN     NaN     NaN   687.0   646.0
    NaN     NaN   687.0   646.0  -189.0
    NaN   687.0   646.0  -189.0  -611.0
  687.0   646.0  -189.0  -611.0  1339.0
  646.0  -189.0  -611.0  1339.0    30.0
 -189.0  -611.0  1339.0    30.0  1645.0
 -611.0  1339.0    30.0  1645.0  -276.0
 1339.0    30.0  1645.0  -276.0   561.0
   30.0  1645.0  -276.0   561.0   470.0
 1645.0  -276.0   561.0   470.0  3395.0
 -276.0   561.0   470.0  3395.0   360.0
  561.0   470.0  3395.0   360.0  3440.0
```

SVR being can only provide single outputs, therefore, to obtain a multi-featured output, the model is wrapped in a Regressor Chain.

The first model in the sequence uses the input and predicts one output; the second model uses the input and the output from the first model to make a prediction; the third model uses the input and output from the first two models to make a prediction, and so on.

For example, if a multioutput regression problem required the prediction of three values y1, y2 and y3 given an input X, then this could be partitioned into three dependent single-output regression problems as follows:

- Problem 1: Given X, predict y1.
- Problem 2: Given X and yhat1, predict y2.
- Problem 3: Given X, yhat1, and yhat2, predict y3.

This can be achieved using the RegressorChain class in the scikit-learn library.

The RNN, LSTM and GRU models each contain a single recurrent segment of 32 units followed by a Dense layer of size 10. They are trained for 30 epochs each in batches of 32.

The Dataset is split into train and test sets in the ratio 3:1. Metric for accuracy is R-Squared:

$$R^2 = 1 - \frac{n \times MSE}{\sum_{i=1}^{n} (y_i - \bar{y})^2}.$$

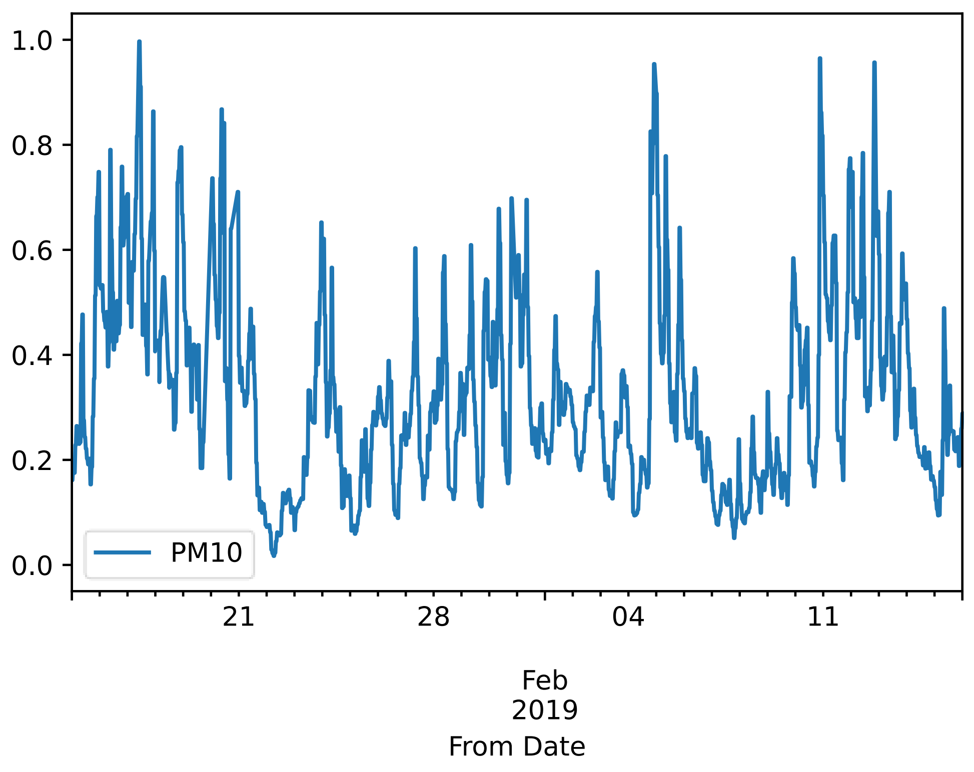
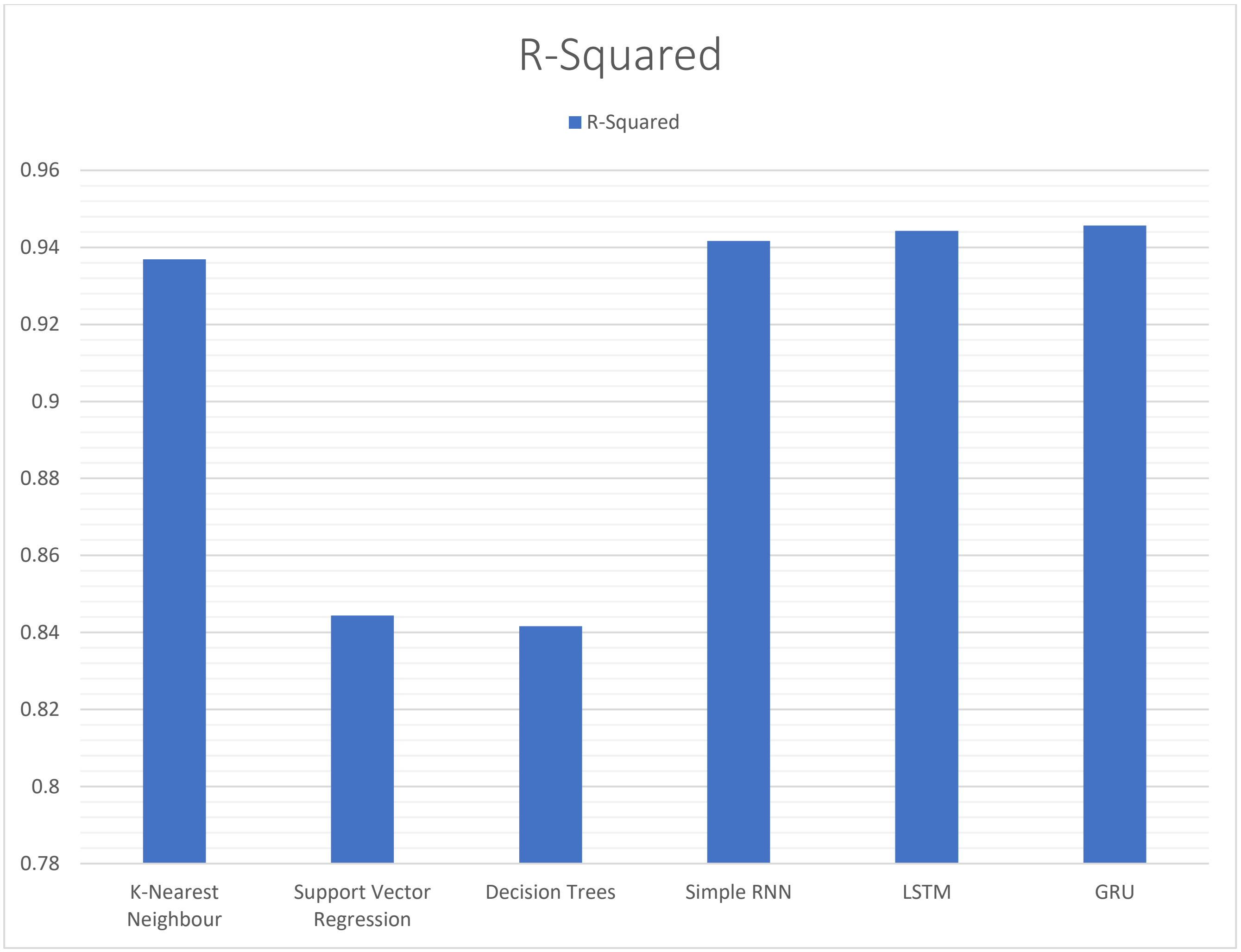


Figure 3 PM10 From 2019-1-15 to 2019-2-16

Method	**R-Squared**
K-Nearest Neighbour	0.9369093648907126
Support Vector Regression	0.8444477518904032
Decision Trees	0.8416144196041311
Simple RNN	0.9416489590067089
LSTM	0.944267206509914
GRU	0.9456603784676176



The RNN, LSTM and GRU models perform consistently better than SVR and D-trees, even without pre-trained weights and only 15 epochs of training.

5. Conclusion and Future Avenues

Machine Learning models, although efficient enough on their own, don’t extract high dimensional information from the data nearly as efficiently as the Deep Learning models. Deep learning is appealing in its potential to improve air quality forecasts in diverse aspects. Its use in air quality forecasting is only in its early stages, and it faces several obstacles.

For starters, there is a data discrepancy that prevents systematic research that lead to breakthroughs in forecasting efficiency improvements. To date, there are no general datasets for testing deep learning algorithms such that benchmarking can be done to gather information for better predictions.

Secondly, given comprehensive datasets, a variety of deep networks and learning algorithms must be compared and improved in order to retrieve complex nonlinear features across scales concealed within such datasets.

Deep networks are often criticized for being opaque and difficult to understand but learning the dynamics of pollutants will allow the underlying deep networks to catch some cause-and-effect interactions as forecasting output improves.

6. References

- Recurrent Neural Networks (RNN) with Keras | TensorFlow Core
- 1.4. Support Vector Machines — scikit-learn 0.24.2 documentation
- 1.6. Nearest Neighbors — scikit-learn 0.24.2 documentation
- 1.10. Decision Trees — scikit-learn 0.24.2 documentation
- Li et al. 2011
- Qi Liao el al. 2020
- Nieto et al. 2013
- Xiang Li et al. 2016
- https://machinelearningmastery.com/feature-selection-time-series-forecasting-python/