International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi :https://doi.org/10.32628/CSEIT2390321

Visibility Analysis and Time Series Forecasting

Asoke Nath*[1], Swapnil Saha[2], Himashri Basu[3], Shalini Torcato[4]
Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India

ARTICLEINFO

Article History:

Accepted: 15 April 2023
Published: 11May 2023

Publication Issue
Volume 9, Issue 3
May-June-2023

Page Number
103-116

ABSTRACT

Visibility prediction and time series forecasting are two important fields in the domain of data analysis and machine learning. Visibility prediction deals with the estimation of atmospheric visibility or the distance over which objects can be clearly seen in a particular location, while time series forecasting involves predicting future values based on historical data. Both these fields have a wide range of applications, including transportation safety, air quality monitoring, and renewable energy management. However, the current prediction of visibility is mostly based on the numerical prediction method similar to the weather prediction [1]. In the present study the authors proposed a method using XGBoost(Extreme Gradient Boosting) and Unsupervised Models(ARIMA, ARMA, LSTM) models to build a weather visibility prediction system. The results obtained were very close to actual dataset.

Keywords : Visibility Prediction, Time Series Forecasting, XGBoost, LSTM

I. INTRODUCTION

In recent years, there has been significant progress in the development of machine learning algorithms that can effectively model and predict visibility and time series data. These algorithms typically use a combination of statistical and machine learning techniques, such as regression analysis, neural networks, and time series analysis. The availability of large datasets and powerful computing resources has also facilitated the development of more sophisticated models and algorithms.

Despite the progress made, there are still several challenges in the field of visibility prediction and time series forecasting, including data quality issues, the need for real-time prediction, and the difficulty of accurately modeling complex and dynamic systems. Addressing these challenges will require further research and development of novel algorithms and techniques.

Overall, visibility prediction and time series forecasting are two critical areas of study that have the potential to inform decision-making in a variety of fields, including transportation, environmental monitoring, and energy management. Ongoing research in these areas will likely lead to new insights and breakthroughs in the coming years.

Visibility is the farthest distance that a person with normal vision can recognize a target from the

background. Visibility information is essential in air quality monitoring which needs to accurate real-time observations of visibility. Severe reduction of visibility often results in great inconveniences to our production and life, and even major traffic accidents. Thus, it is of great significance to establish an accurate forecast visibility system. To satisfy the needs for accurately measuring visibility, many models have emerged to solve the problem[2] .Modern application on real-time data can generalize on the predicted data. So, it is reasonable to use the real-time data to obtain the correlation between visibility and possible causes. Thus, the authors focus on exploring a viable solution to build a highly accurate visibility prediction system[1]. Traditional methods [5][6] of visibility prediction focus on the correlation between meteorological factors and visibility. With the rise of machine learning, many visibility prediction schemes using machine learning have been proposed in recent years [3][4][10][11]. These methods use large amounts of meteorological data to search the latent relation between the input and visibility, and then applied it to visibility prediction.[1]

Time series forecasting occurs when the researchers make scientific predictions based on historical time stamped data. It involves building models through historical analysis and using them to make observations and drive future strategic decision-making. An important distinction in forecasting is that at the time of the work, the future outcome is completely unavailable and can only be estimated through careful analysis and evidence-based priors.

Time series forecasting is the process of analyzing time series data using statistics and modeling to make predictions and inform strategic decision-making. It's not always an exact prediction, and likelihood of forecasts can vary wildly-especially when dealing with the commonly fluctuating variables in time series data as well as factors outside our control. However, forecasting insight about which outcomes are more likely—or less likely—to occur than other potential outcomes. Often, the more comprehensive the data we have, the more accurate the forecasts can be.[21]

It is **extremely difficult to get hold of real-time data** for the supervised model to make predictions**.** So therefore, in the present study the authors intend to harvest the applications of both supervised and unsupervised learning to predict the **future visibility**. In supervised learning algorithm the authors have used an XGBoostRegressor model, which will predict the visibility conditions based on the input parameters(humidity, temperature, wind speed, pressure, etc) and provides an estimate of the visibility level in a given area.

The unsupervised learning models on the other hand will be used to predict the parameters(humidity, temperature, wind speed, pressure, etc) of the future. The authors **connect** the **unsupervised and supervised models** in such a manner that the output of the unsupervised model will be the input for the supervised model, which will in turn give a simulation for real-time visibility analysis.

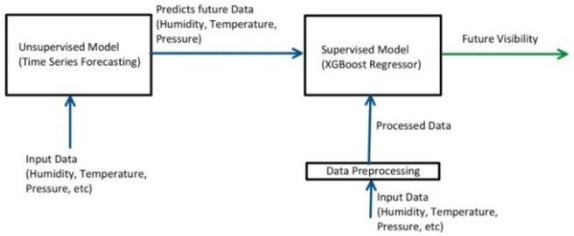


Fig-1: Block Diagram of Unsupervised and Supervised Model

II. LITERATURE REVIEW

Visibility is the farthest distance that a person with normal vision can recognize a target from the background. Visibility information is essential in air quality monitoring which needs to accurate real-time observations of visibility. Severe reduction of visibility often results in great inconveniences for production and life, and even major traffic accidents. Thus, it is of great significance to establish an accurate forecast visibility system[1]. To satisfy the needs for accurately

measuring visibility, many models have emerged to solve the problem [2]. The works in [3] [4] indicated that the application on real-time data can generalize on the predicted data. So it is reasonable to use the real-time data to research the correlation between visibility and possible causes.

A. Ensemble techniques

Ensemble techniques are a popular approach to machine learning (ML) that involve combining the predictions of multiple models to improve overall performance. There are several types of ensemble techniques in ML, including:

Bagging: This technique involves creating multiple independent models, each trained on a different subset of the training data. The predictions of these models are then averaged to obtain a final prediction.

Bagging (Bootstrap Aggregating) is an ensemble learning method that combines multiple models to improve the overall prediction accuracy. The bagging algorithm can be described mathematically as follows:

Let there be a training dataset D of size N, where each sample is represented as a tuple (x, y) where x is the input feature vector and y is the corresponding output label.

Generate B bootstrap samples D1, D2, ..., DB of size N by randomly sampling from D with replacement.For each bootstrap sample Di, train a model Mi on Di using a chosen learning algorithm.

After training all B models, the final prediction is obtained by aggregating the predictions of each model. In classification problems, this can be done by taking the majority vote of the predicted class labels. In regression problems, this can be done by taking the average of the predicted values.

Overall, the bagging algorithm can improve the generalization performance of a model by reducing the variance of the predictions and increasing the stability of the model.

Boosting: Boosting is a technique that combines several weak models into a single strong model. The models are created iteratively, with each new model focusing on the examples that were incorrectly classified by the previous models.Boosting is a machine learning technique that combines several weak learners to form a strong learner. The basic idea behind boosting is to iteratively train a series of weak learners on weighted versions of the training data, where the weights are adjusted after each iteration to focus on the samples that were misclassified by the previous learner. The final prediction is then made by taking a weighted average of the predictions of the weak learners.

The basic idea behind AdaBoost,which is one of the most popular boosting algorithms, is to give higher weights to the misclassified samples and lower weights to the correctly classified samples, so that the next weak learner focuses more on the difficult samples. The learning rate controls the contribution of each weak learner to the final hypothesis, and it depends on the error of the weak learner. The final hypothesis combines the predictions of all the weak learners using their respective learning rates.

Gradient Boosting is a machine learning technique that uses an ensemble of decision trees to create a powerful model. Like other boosting algorithms, gradient boosting builds an ensemble of weak learners, but it differs from AdaBoost in the way it updates the weights and combines the weak learners.

The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function. Explicit regression gradient boosting algorithms were subsequently developed, by Jerome H. Friedman, simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean. The latter two papers introduced the view of boosting algorithms as iterative functional gradient descent algorithms. That is, algorithms that optimizes a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development

of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

Gradient Boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees. When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest.

The main idea behind gradient boosting is to fit each weak learner to the residual error of the previous weak learners, so that the ensemble progressively improves the prediction performance. More specifically, given a training dataset S = {(x1, y1), (x2, y2), ..., (xn, yn)}, where xi is a d-dimensional input and yi is a scalar output, gradient boosting aims to find a function F(x) that minimizes the mean squared error (MSE) loss over the training data.

To prevent overfitting, gradient boosting also uses regularization techniques, such as shrinkage and early stopping. Shrinkage reduces the learning rate α, so that each new weak learner has less impact on the final prediction, while early stopping stops the boosting process when the validation error no longer improves, indicating that further iterations may only increase overfitting.

Overall, gradient boosting is a powerful algorithm that can fit complex functions to large datasets, and is widely used in various machine learning tasks, such as regression, classification, and ranking.

Ensemble techniques can help to improve the accuracy and robustness of machine learning models. However, they can also be computationally expensive and require careful tuning of hyperparameters to achieve optimal performance.

Root Mean Squared error (RMSE): Root mean squared error (RMSE) is a common evaluation metric used in machine learning to measure the accuracy of a regression model. RMSE is a measure of the average deviation between the predicted values and the true values. It is calculated as the square root of the average of the squared differences between the predicted and actual values:

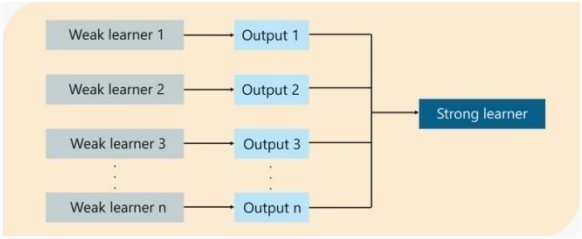
$$\text{RMSE} = \sqrt{\left(\frac{1}{n}\right) * \left(\sum_{i=1}^{n} (yi - \hat{y}i)\text{^}2\right)}$$

where n is the number of samples, yi is the true value, and ŷi is the predicted value. The square of the differences (yi - ŷi)^2 is taken to ensure that both overestimation and underestimation contribute to the error equally.

RMSE is a useful metric because it penalizes larger errors more heavily than smaller errors. This is important because larger errors can have a greater impact on the overall performance of the model.

In addition to RMSE, other metrics such as mean absolute error (MAE) and R-squared (R2) are also commonly used in regression tasks. MAE measures the average absolute difference between the predicted and actual values, while R2 measures the proportion of the variance in the target variable that can be explained by the model. The choice of evaluation metric depends on the specific problem and the nature of the data.

Extreme Gradient Boost(XGBoost) initially started as a research project by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC) group. Initially, it began as a terminal application which could be configured using a libsvm(a standard Machine Learning Library) configuration file. It became well known in the ML competition circles after its use in the winning solution of the Higgs Machine Learning Challenge. Soon after, the Python and R packages were built, and XGBoost now has package implementations for Java, Scala, Julia, Perl, and other languages. This brought the library to more developers and contributed to its popularity among the Kaggle community, where it has been used for a large number of competitions.



Source: [22]

Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler and weaker models. The XGBoost algorithm performs well in machine learning competitions because of its robust handling of a variety of data types, relationships, distributions, and the variety of hyperparameters that you can fine-tune.XGBoost is a more regularized form of Gradient Boosting. XGBoost uses advanced regularization (L1 & L2), which improves model generalization capabilities.XGBoost delivers high performance as compared to Gradient Boosting. Its training is very fast and can be parallelized across clusters.

The **XGBoost classifier** can be mathematically described as follows:

1. Let {x_i, y_i} be the training data where x_i represents the features of the ith observation, and y_i represents the target variable (0 or 1 for binary classification).
2. The goal of the XGBoost algorithm is to learn a function F(x) that maps the features to the target variable. This function is represented as an ensemble of K decision trees:F(x) = f_1(x) + f_2(x) +….+ f_k(x)where f_k(x) represents the kth decision tree.
3. The decision tree f_k(x) is constructed recursivel by partitioning the feature space into smaller an smaller regions. At each node of the tree, a binar split is made on a single feature that maximizes splitting criterion. This splitting criterion i typically the reduction in impurity, which i measured using a loss function L(y_i, F(x_i)) that compares the predicted output F(x_i) to the actual output y_i.
4. The XGBoost algorithm uses a gradient boosting approach to iteratively improve the performance of the decision tree ensemble. At each iteration t, a new decision tree f_t(x) is trained to minimize the following objective function:obj^(t)= $\sum_{i=1}^{n}$ L(y_i, F^(t − 1)(x_i) + f_t(x_i)) + Ω(f_t)where,
   - obj^(t) represents the objective function to be optimized in the t-th iteration of a boosting algorithm.
   - $\sum_{i=1}^{n}$. represents the sum over all training examples i=1,2,…,n.
   - L(y_i, F^(t-1)(x_i) + f_t(x_i)) is the loss function that measures the difference between the true label y_i and the predicted value F^(t-1)(x_i) + f_t(x_i) for the i-th example, where F^(t-1)(x_i) is the prediction made by the ensemble of trees at iteration t-1, and f_t(x_i) is the prediction made by the new tree being added at iteration t.
   - Ω(f_t) is the regularization term that penalizes complex trees, which is often expressed as a sum of the weights of the individual leaves in the tree or as a function of the depth of the tree. The purpose of this term is to prevent overfitting of the model.
5. The objective function is optimized using a greedy search algorithm that adds one tree at a time to the decision tree ensemble. The search algorithm selects the best split for each node of the tree by computing the gradient and Hessian of the loss function with respect to the predicted output F^{(t-1)}(x_i).
6. Finally, the XGBoost algorithm outputs the function F(x) that is the sum of all the decision trees in the ensemble. The predicted output for a new observation x is computed by traversing each decision tree and summing the output values of the leaf nodes that x falls into.

In summary, the XGBoost classifier uses an ensemble of decision trees that are trained using a gradient boosting approach to minimize a regularized objective function that measures the discrepancy between the predicted output and the actual output. The classifier outputs a function that maps the features to the target variable by summing the outputs of all the decision trees in the ensemble.

The **XGBoostregressor** can be defined mathematically as follows:

Given a training set of input-output pairs, (x_1, y_1), (x_2, y_2), …, (x_n, y_n), where $x_i \in \mathbb{R}^d$ is a d-dimensional input vector and $y_i \in \mathbb{R}$ is the corresponding output, the XGBoost algorithm aims to learn a function f(x) that maps inputs to outputs.

To do this, XGBoost starts with an initial estimate of f(x), denoted as $f_0(x)$, which is typically set to the mean of the target values in the training set. It then iteratively refines this estimate by adding new weak learners to the model.

At each iteration t, XGBoost trains a new decision tree, denoted as $h_t(x)$, to predict the residuals between the true output and the current estimate of the output:

$r_i = y_i - f_{t-1}(x_i)$

where $f_{t-1}(x_i)$ is the output of the model at iteration t1.

The new tree $h_t(x)$ is trained to minimize the mean squared error (MSE) of the residuals

$L(h) = \sum_i (r_i - h(x_i))^2$

where the sum is over all training examples.

To prevent overfitting, XGBoost regularizes the tree by adding penalties for complexity and size. This is done by adding a regularization term to the loss function:

$L(h) = \sum_i (r_i - h(x_i))^2 + \gamma T(h) + \lambda ||w||^2$

where T(h) is a measure of the complexity of the tree, w is the vector of leaf weights, and γ and λ are regularization parameters.

Finally, XGBoost combines the new tree with the previous estimate of the function f(x) using a learning rate η to control the contribution of each new tree:

$f_t(x) = f_{t-1}(x) + \eta h_t(x)$

This process is repeated for a fixed number of iterations or until the loss function converges. The final model is the sum of all the trees:

$f(x) = \sum_t f_t(x)$

where the sum is over all the trees in the model.

In summary, the XGBoost algorithm learns a function f(x) by iteratively adding decision trees to the model, each trained to predict the residuals between the true output and the current estimate of the output. Regularization is used to prevent overfitting, and a learning rate is used to control the contribution of each tree. The final model is the sum of all the trees in the ensemble.

B. Time series forecasting

Time series forecasting is a branch of machine learning that involves predicting future values of a time-dependent variable based on its past behavior. In this type of problem, the data points are collected over time and are therefore correlated. Time series forecasting is commonly used in finance, economics, weather forecasting, and many other fields where the accurate prediction of future values is crucial.

There are many machine learning techniques that can be used for time series forecasting, including:

Autoregressive(AR) Models: In a multiple regression model, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of past values of the variable. The term autoregression indicates that it is a regression of the variable against itself.We refer to this as an AR(p) model, an autoregressive model of order p.

Moving Average(MA) Models:In time series analysis, the moving-average model (MA model), also known as moving-average process, is a common approach for modeling univariate time series. The moving-average model specifies that the output variable is cross-correlated with a non-identical to itself random-variable.

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model.

ARMA(autoregressive–moving-average): The autoregressive (AR) component of the model is the first summation term, which models the current value of the time series as a linear combination of its past values, weighted by the autoregressive parameters c(i). The moving average (MA) component of the model is the second summation term, which models the current value of the time series as a linear combination of its past errors, weighted by the moving average parameters θ(j). The error term ε(t)represents the random noise in the data that cannot be explained by the AR and MA components.

The ARMA model is typically denoted by ARMA(p, q), where p is the order of the autoregressive component (i.e., the number of lagged values of the time series used to predict the current value), and q is the order of the moving average component (i.e., the number of lagged errors used to predict the current value). ARMA models are widely used in time series analysis and forecasting, and their parameters can be estimated using various statistical techniques, such as maximum likelihood estimation or Bayesian inference.

ARIMA(autoregression integrated moving average)ARIMA (Autoregressive Integrated Moving Average) is a mathematical model used for analyzing and forecasting time series data. It is a combination of three models: Autoregression (AR), Integrated (I), and Moving Average (MA).

The AR model represents the linear dependence of the current observation on one or more past observations. It assumes that the current value of the time series can be predicted by a linear combination of its past values. The order of AR model is represented by 'p' which is the number of lagged observations used to predict the current observation.

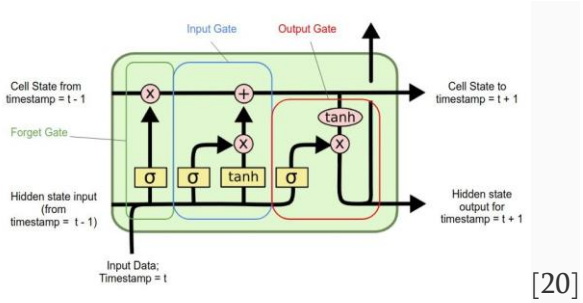
The 'I' component represents the degree of differencing required to make the time series stationary. Differencing involves subtracting the current observation from its previous observation, which is used to remove any trend or seasonality from the time series.

The ARIMA model is denoted by ARIMA(p,d,q) where 'p' represents the order of the AR component, 'd' represents the degree of differencing required to make the time series stationary, and 'q' represents the order of the MA component.

LSTM(Long short-term memory)(Recurrent Neural Network):

Long short-term memory (LSTM)[2] is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network (RNN) can process not only single data points (such as images), but also entire sequences of data (such as speech or video). This characteristic makes LSTM networks ideal for processing and predicting data.

The name of LSTM refers to the analogy that a standard RNN has both "long-term memory" and "short-term memory". The connection weights and biases in the network change once per episode of training, analogous to how physiological changes in synaptic strengths store long-term memories; the activation patterns in the network change once per time-step, analogous to how the moment-to-moment change in electric firing patterns in the brain store short-term memories.[5] The LSTM architecture aims to provide a short-term memory for RNN that can last thousands of timesteps, thus "long short-term memory".



[20]

Fig: LSTM cell

The mathematical model of an LSTM cell consists of several equations that govern its behavior. Let's consider a single LSTM cell with an input sequence

X = (x1, x2, ..., xn),

where xt is the input at time step t, and a hidden state sequence

H = (h1, h2, ..., hn),

where ht is the hidden state at time step t.

The LSTM cell also has an output sequence

Y = (y1, y2, ..., yn), where yt is the output at time step t.

The LSTM cell contains several gates that control the flow of information into and out of the cell. These gates are defined as follows:

Forget gate: This gate decides which information from the previous hidden state should be forgotten. The forget gate is computed as follows:

$$ft = \sigma(Wf \cdot [ht-1, xt] + bf)$$

where σ is the sigmoid activation function, Wf is the weight matrix, [ht-1, xt] is the concatenation of the previous hidden state and the current input, and bf is the bias term.

Input gate: This gate decides which information from the current input should be stored in the cell state. The input gate is computed as follows:

$it = \sigma(Wi \cdot [ht-1, xt] + bi)$

$ct = \tanh(Wc \cdot [ht-1, xt] + bc)$

$Ct = ft \odot Ct-1 + it \odot ct$

where ⊙ denotes element-wise multiplication, tanh is the hyperbolic tangent activation function, Wi, Wc are weight matrices, bi, bc are bias terms, and Ct-1 is the previous cell state.

Output gate: This gate decides which information from the cell state should be outputted as the hidden state. The output gate is computed as follows:

$ot = \sigma(Wo \cdot [ht-1, xt] + bo)$

$ht = ot \odot \tanh(Ct)$

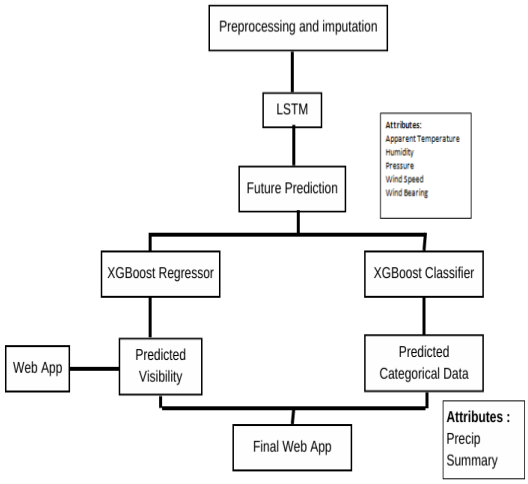
where Wo is the weight matrix, bo is the bias term, and ⊙ and tanh are defined as before.

The LSTM cell can be extended to multiple layers by stacking multiple cells on top of each other. Each layer receives the hidden state sequence from the previous layer as input. The output of the last layer is usually used for prediction or classification tasks.

Note that the LSTM cell can also be modified and extended in various ways, such as adding peephole connections or using different activation functions.

III. ALGORITHM OF PROPOSED STUDY

The detailed block diagram of the project models



The proposed algorithm undergoes the following methods:

A. DATA PREPROCESSING

- Formatting the date string: The original dataset had date and time stamp integrated in one single index column.
- Data imputation: SimpleImputer is a Python library that provides various methods for imputing missing values in a dataset. It can be used to fill in missing data with mean, median, mode or constant values.
- One hot encoding is a data transformation technique used to convert categorical variables into a format that can be easily processed by machine learning algorithms.
- Splitting data into Training and testing sets: Splitting data into training and testing sets using sklearn.preprocessing.train_test_split. The basic idea is to use part of the available data to train the model, and another part to test the model's performance.

B. TIME SERIES FORECASTING

ARIMA(Auto Regression Integrated Moving Average) model was initially used to train and predict the numerical data. It turned out to be inefficient in terms of time and accuracy.

So, LSTM was used to predict the numerical data.

LSTM : The Long Short Term Memory(LSTM) is a layer network each layer contains several LSTM cells, which are connected to the corresponding cells in the adjacent layers. The input to the network is fed into the first layer, and output of the last layer is the final prediction. Unlike standard feedforward neural networks, LSTM has feedback connections.

Given below is a snippet for the LSTM model(used for Time series forecasting of the numerical variables). This is implemented using TimeSeriesGenerator, which helps in making batches of inputs and outputs for the Recurrent Neural Network.

```
from keras.preprocessing.sequence import TimeseriesGenerator
n_input = 8
n_features= 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size=1)
```

```
a,b=generator[0]
print(f'Given the Array: \n{a.flatten()}')
print(f'Predict this y: \n {b}')
```

```
Given the Array:
[0.97013513 0.97061297 0.97090923 0.9713584  0.97145397 0.97159732
 0.97165466 0.97176934]
Predict this y:
 [[0.97227585]]
```

C. PREDICTING VISIBILITY

DMatrix is specifically designed for use with the XGBoost algorithm, Using DMatrix with XGBoost, we can improve the overall performance of the model and make the code more readable and maintainable.

Hyperparameter Tuning

We use the inbuilt xgboost function xgboost.cv for hyperparameter tuning. This use cross validation at its core and the use of DMatrix improves efficiency of this cross validation.

```
gridsearch_params = [
    (max_depth, min_child_weight)
    for max_depth in range(4,12)
    for min_child_weight in range(1,4)
]
```

```
CV with max_depth=10, min_child_weight=3
        MAE 2.287653727175157 for 8 rounds
CV with max_depth=11, min_child_weight=1
        MAE 2.2593444239839893 for 56 rounds
CV with max_depth=11, min_child_weight=2
        MAE 2.2562014539962036 for 73 rounds
CV with max_depth=11, min_child_weight=3
        MAE 2.2792271121675163 for 8 rounds
Best params: 11, 2, MAE: 2.2562014539962036
```

The visibility is being predicted for the set of values of temperature, apparent temperature, humidity, pressure, wind speed and wind bearing. After tuning the model for quite a few number of times the final minimum root mean squared error comes as follows:

```
mean_absolute_error(best_model.predict(dtest), y_test_num)
```

```
2.19630285331274
```

D. PREDICTING CATEGORICAL DATA

The numerical data(Temperature, Pressure, Humidity, etc) are used to train the Xgboost Classifier which preditcts the Categorical Variables namely- Precipitation Type and Summary

Label Encoding:

Label encoding is a process of converting categorical data into numerical form. In this process, each unique category or label in a categorical variable is assigned a numerical value. The numerical values assigned to each label are arbitrary and depend on the order of the labels in the dataset.

```
fromsklearn.preprocessingimportLabelEncoder
le = LabelEncoder()
df_imp['Daily Summary'] =
le.fit_transform(df_imp['Daily Summary'])
df_imp['Summary'] =
le.fit_transform(df_imp['Summary'])
df_imp['Precip Type'] =
le.fit_transform(df_imp['Precip Type'])
```

Hyperparameter Tuning

Hyperparameter tuning of these classification models is done using RandomizedSearchCV where a fixed number of parameter settings is sampled from the specified distributions. The best hyperparameters are as follows:

```
params1={"learning_rate" : [0.05, 0.07, 0.10, 0.12, 0.15, 0.17, 0.20, 0.22, 0.25, 0.30 ],
        "max_depth" : [ 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16],
        "min_child_weight" : [ 1, 2, 3, 4, 5, 6, 7 ],
        "gamma" : [0.0, 0.1, 0.2 ,0.25, 0.3, 0.32, 0.35, 0.4, 0.5 ],
        "colsample_bytree" : [ 0.1,0.2, 0.25, 0.3, 0.4, 0.45, 0.5 , 0,6, 0.7 ]}
classifier=xgboost.XGBClassifier()
random_search=RandomizedSearchCV(classifier,param_distributions=params1,n_iter=5,
                                 scoring='accuracy',n_jobs=-1,cv=5,verbose=3)
```

```
random_search.best_estimator_
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.4, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.2, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.15, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=14, max_leaves=None,
              min_child_weight=6, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

IV. RESULTS AND DISCUSSIONS

The prediction of the numerical data using LSTM Time Series Generator are as follows:

A. TEMPERATURE

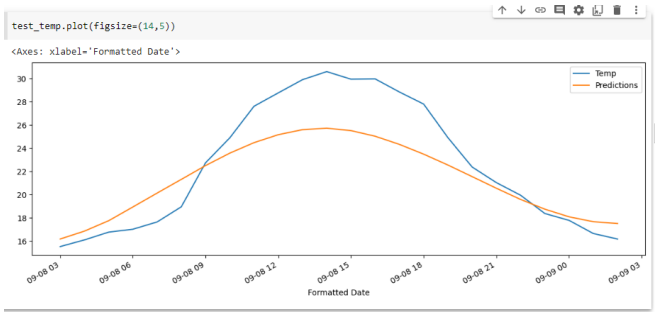


Fig 1. Graph of test[‘Temp’] vs Prediction

B. APPARENT TEMPERATURE

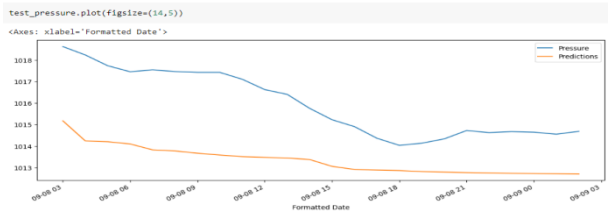


Fig 2. Graph of test[‘Apperent Temp’] vs Prediction

C. HUMIDITY

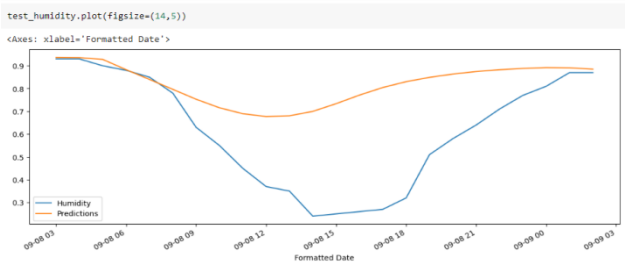


Fig 3. Graph of test[‘Humidity’] vs Prediction

D. PRESSURE

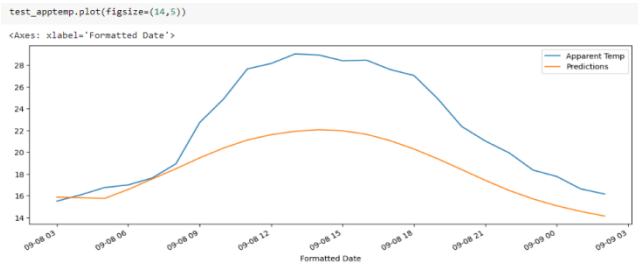


Fig 4. Graph of test[‘Pressure’] vs Prediction

E. WIND SPEED

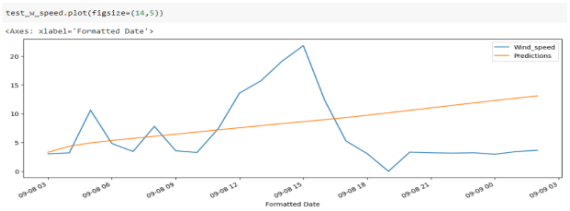


Fig 5. Graph of test[‘Wind_speed’] vs Prediction

F. WIND BEARING

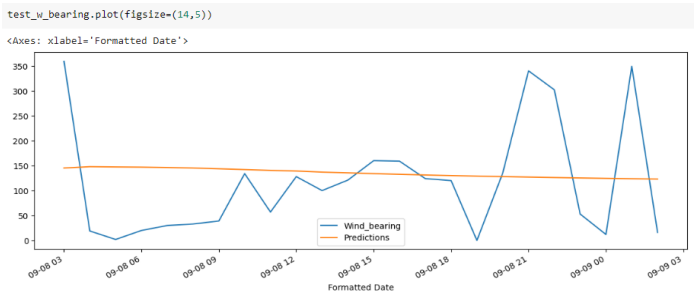


Fig 6.Graph of test[‘Wind_bearing’] vs Prediction

After converting the categorical data, i.e., the attributes: precip type, summary into numerical form using label encoding, the model is optimized using RandamoziedSearchCV. After training the model, predictions are made using test data(rain and

summary), to get the following classification reports and confusion matrices.PRECIP TYPE:

```
acc 1.0
f1              precision    recall  f1-score   support

           0       1.00      1.00      1.00     17201
           1       1.00      1.00      1.00      2090

    accuracy                           1.00     19291
   macro avg       1.00      1.00      1.00     19291
weighted avg       1.00      1.00      1.00     19291
```

Fig 7.a. Classification Report of Precip Type

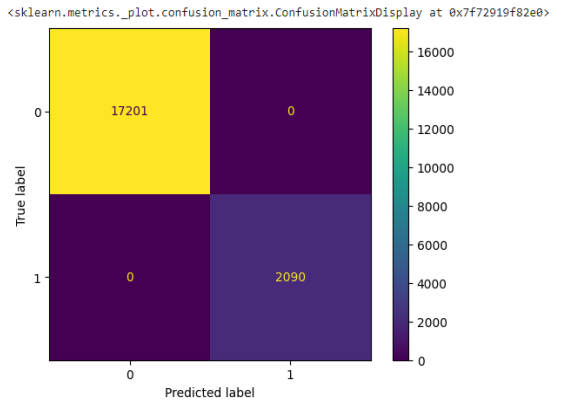


Fig 7b.Confusion Matrix of Precip Type

The above output shows that the modelgives 100% accuracy for the attribute “precip type”.

SUMMARY

```
acc 0.6744077549116169
f1              precision    recall  f1-score   support

           0       0.62      0.62      0.62         8
           2       0.71      0.83      0.77         6
           3       0.58      0.59      0.59       105
           4       0.69      0.66      0.67       103
           5       0.69      0.70      0.70        67
           6       0.71      0.62      0.66      2177
           8       1.00      0.12      0.22         8
           9       1.00      0.33      0.50         6
          10       0.00      0.00      0.00         3
          11       0.33      0.10      0.15        21
          12       1.00      1.00      1.00      1421
          13       0.33      0.67      0.44         3
          14       0.00      0.00      0.00         2
          15       0.00      0.00      0.00         4
          16       0.60      0.50      0.55        12
          17       0.59      0.59      0.59      5570
          18       0.69      0.62      0.65      3347
          19       0.66      0.73      0.70      6393
          20       1.00      0.67      0.80         3
          21       0.00      0.00      0.00         4
          23       0.00      0.00      0.00         1
          24       0.33      0.40      0.36         5
```

Fig 8.a. Classification Report of Summary

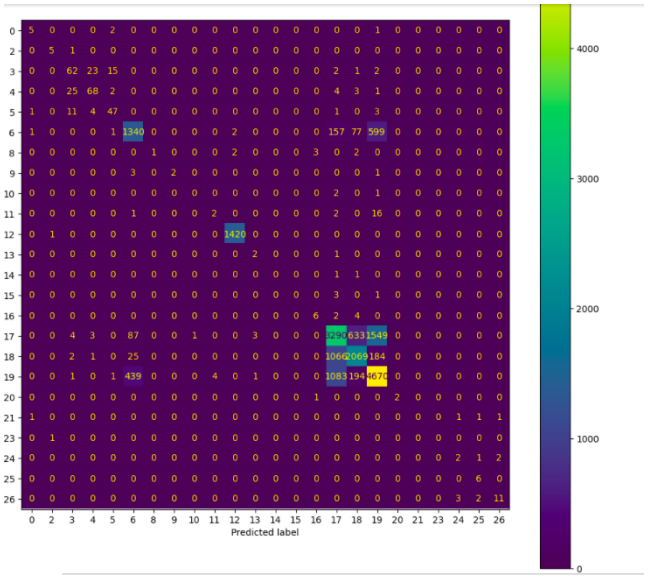


Fig 8b.Confusion Matrix of Summary

The above output shows that the model gives 67.44% accuracy for the attribute “summary”.The outputs of the lstm are fed as input to the visibility model created earlier to get the following results

```
dtest= xgb.DMatrix(df_pred)
predv = model3.predict(dtest)
predv
```

```
array([ 9.449733 ,  6.9187202,  7.3552637, 12.047621 , 14.325575 ,
       15.147667 , 15.215871 , 13.440252 , 11.614527 , 11.55043  ,
       11.564219 , 11.476882 , 11.7572775, 11.064631 , 11.234624 ,
       11.662162 , 12.571036 , 13.190052 , 13.61522  , 12.675921 ,
       10.351802 , 10.760644 , 10.203035 , 11.044655 ], dtype=float32)
```

```
dtest= xgb.DMatrix(df_main)
testv = model3.predict(dtest)
testv
```

```
array([13.698466, 11.788316, 13.582067, 11.799308, 13.488463, 11.666311,
       12.884146, 12.810701, 12.274558, 12.04324 , 12.221408, 12.142826,
       11.67698 , 11.60242 , 12.130083, 12.371131, 13.569483, 12.793567,
       12.972131, 14.129662, 15.124175, 14.354009, 14.582872, 12.876717],
      dtype=float32)
```

```
mse= mean_squared_error(testv,predv)
rmse=math.sqrt(mse)
rmse
```

```
2.6165784440283195
```

V. CONCLUSION AND FUTURE SCOPE

Visibility analysis is a technique used in various fields, including architecture, urban planning, transportation engineering, and military operations, to assess the visibility of objects or areas from a particular location or perspective. With advances in technology and computing power, the future scope of visibility analysis is expected to expand in several ways, including:

Integration with augmented reality and virtual reality: Visibility analysis can be integrated with augmented and virtual reality technologies to create immersive and interactive environments. For example, architects can use visibility analysis to create virtual models of buildings that can be explored from different angles and viewpoints.

FutureScope: Use of drones and aerial imagery: Drones and aerial imagery can be used to capture high-resolution images of terrain and buildings, which can be used for visibility analysis. This technology can be used for a range of applications, such as urban planning and military operations.

Real-time analysis: With the development of real-time rendering technologies, visibility analysis can be performed in real-time, allowing for dynamic changes in the environment to be taken into account. This can be useful for applications such as traffic management and emergency response planning.

Integration with machine learning: Machine learning algorithms can be used to improve the accuracy and speed of visibility analysis. For example, machine learning algorithms can be used to automatically identify objects or features that are likely to obstruct visibility.

Overall, the future scope of visibility analysis is promising, with new technologies and techniques expected to enhance the accuracy and usefulness of this technique in a range of applications. The future scope of time series analysis is vast and diverse, with potential applications in a wide range of industries. With the increasing availability of data, it is expected to become even more critical in the coming years.

VI. REFERENCES

[1]. Chuang Zhang, Ming Wu, Jinyu Chen and Kaiyan Chen, "Weather Visibility Prediction Based on Multimodal Fusion".Article in IEEE Access –June 2019

[2]. Y. He, X. Sun, L. Gao, and B. Zhang, "Ship detection without sea-landsegmentation for large-scale high-resolution optical satellite images," 07 2018, pp. 717–720.

[3]. Z. W. Wang, C. L. Zhang, C. Su, and C. L. Cheng, "On modeling ofatmospheric visibility classification forecast with nonlinear support vectormachine," in International Conference on Natural Computation, 2009.

[4]. R. F. Chevalier, G. Hoogenboom, R. W. Mcclendon, and J. A. Paz,"Support vector regression with reduced training sets for air temperatureprediction: a comparison with artificial neural networks," Neural Computing and Applications, vol. 20, no. 1, pp. 151–159, 2011.

[5]. C. Marzban, S. Leyton, and B. Colman, "Ceiling and visibility forecastsvia neural networks," Weather & Forecasting, vol. 22, no. 3, pp. 466–479,2006.

[6]. Allan C. Just, Yang Liu, MeytarSorek-Hamer, Johnathan Rush, Michael Dorman, Robert Chatfield, Yujie Wang, Alexei Lyapustin, ItaiKloog, "Gradient Boosting Machine Learning to Improve Satellite-Derived Column Water Vapor Measurement Error"

[7]. Jeongmook Park, Yongkyu Lee, and Jungsoo Lee, Researcher, Forest ICT Research Center, National Institute of Forest Science Department of Forest Management, Kangwon National University, Chuncheon, Republic of Korea-" Assessment of Machine Learning Algorithms for Land Cover Classification Using Remotely Sensed Data"

[8]. "Nonlinear analysis and visibility forecast of the relation between atmospheric visibility and pm2.5 concentration and relative humidity inwuhan," Journal of Meteorology, vol. 74, no. 2, 2016.

[9]. R. M. Chmielecki and A. E. Raftery, "Probabilistic visibility forecastingusing bayesian model averaging," Monthly Weather Review, vol. 139,no. 5, pp. 1626–1636,2016.

[10]. D. Bari, "Visibility prediction based on kilometric nwp model outputsusing machine learning regression," in IEEE 14th International Conference on e Science, 2018.
[11]. Z. Ma, J. H. Xue, A. Leijon, Z. H. Tan, Z. Yang, and J. Guo, "Decorrelationof neutral vector variables: Theory and applications," IEEE Transactionson Neural Networks and Learning Systems, vol. PP, no. 99, pp. 1–15,2016.
[12]. W. Kai, Z. Hong, A. Liu, and Z. Bai, "The risk neural network basedvisibility forecast," in International Conference on Natural Computation,2009.
[13]. C. Marzban, S. Leyton, and B. Colman, "Ceiling and visibility forecastsvia neural networks," Weather & Forecasting, vol. 22, no. 3, pp. 466–479,2006.
[14]. K. Abhishek, M. Singh, S. Ghosh, and A. Anand, "Weather forecastingmodel using artificial neural network," Procedia Technology, vol. 4,pp. 311 – 318, 2012, 2nd International Conference on Computer,Communication, Control and Information Technology(C3IT-2012) onFebruary 25 - 26, 2012. [Online].Available: http://www.sciencedirect.com/science/article/pii/S221201731200326X
[15]. Y. T. Tsai, Y. R. Zeng, and Y. S. Chang, "Air pollution forecasting using rnn with lstm," in 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology-Congress,2018.
[16]. H. Tian, K. Li, and M. Bo, "A novel prediction modeling scheme based on multiple information fusion for day-ahead electricity price," in Chinese Control and Decision-Conference,2011.
[17]. H. Chen and S. Huang, "A comparative study on model selection and multiple model fusion," in International Conference on Information Fusion,2005
[18]. M. Sheng, K. Hu, G. Hao, and X. Wang, "Small fire smoke region location and recognition in satellite image," in International Congress on Image & Signal Processing,2017.
[19]. S. Miao, H. Lin, H. Gao, and L. Dong, "Strip smoke and cloud recognition in satellite image," in International Congress on Image & Signal Processing, 2017.
[20]. https://www. turing.com/kb/comprehensive-guide-to-lstm-rnn
[21]. https://www.tableau.com/learn/articles/time-series-forecasting#:~:text=Time%20series%20forecasting%20occurs%20when,drive%20future%20strategic%20decision%2Dmaking
[22]. https://www.edureka.co/blog/boosting-machine-learning
[23] Dataset source: https://www.kaggle.com/datasets/muthuj7/weather-dataset

Cite This Article :

Asoke Nath, Swapnil Saha, Himashri Basu, Shalini Torcato, "Visibility Analysis and Time Series Forecasting", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 9, Issue 3, pp.103-116, May-June-2023. Available at doi : https://doi.org/10.32628/CSEIT2390321

Journal URL : https://ijsrcseit.com/CSEIT2390321

AUTHOR'S PROFILE

Dr. Asoke Nath is working as an Associate Professor in the Department of Computer Science, St. Xavier's College (Autonomous). Kolkata. He is engaged in research work in the field of Cryptography and Network Security, Steganography, Green Computing,Big data analytics, Li-Fi Technology, Mathematical modelling of Social Area Networks, MOOCs, Quantum Computing etc. He has published more than 269 research articles in different Journals and conference proceedings.

Mr. Swapnil Saha is currently a 3rd year undergraduate student in St. Xavier's College(Autonomous), Kolkata and has a diploma in Data Science from Indian Institute of Technology(IIT), Madras. His interests include Machine Learning, big data analysis and Cryptography.

Ms. Himashri Basu is currently a 3rd year UG student in the Department of Computer Science, St. Xavier's College (Autonomous), Kolkata. Her interests lie in Software Engineering, Machine Learning, Web Development, and their real-world implementations.

Ms. Shalini Torcato is currently a 3rd year undergraduate student in the Department of Computer Science ,St. Xavier's College (Autonomous), Kolkata .Her interests lie in Web designing , Cryptography and Network Security.