VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

SEMESTER 241 - CO3117

MACHINE LEARNING ASSIGNMENT

Air Pressure Prediction

Instructor: **PROF. Nguyen Duc Dung**

Le Van Phuc	- 2152241
Nguyen Quang Thien	- 2152994

HO CHI MINH CITY, JANUARY 2025

Contents

1 Introduction

1.1 Problem Statement

Atmospheric pressure plays a vital role in meteorological studies, influencing weather patterns, climate systems, and forecast accuracy. Precise pressure estimation is essential for aviation, agriculture, and environmental science applications. Traditional forecasting methods rely heavily on physical models, which, while robust, often demand significant computational resources and expertise. With the advent of machine learning, data-driven approaches have emerged as promising alternatives for predicting pressure using large-scale meteorological datasets.

1.2 Motivation

Atmospheric pressure estimation is crucial for weather prediction, storm tracking, and understanding atmospheric dynamics. Traditional numerical weather models, while comprehensive, are resource-intensive and limited by their dependence on precise physical parameterizations. Machine learning provides a complementary approach, offering rapid and accurate predictions without the explicit need for physical equations.

This study explores machine learning models for estimating atmospheric pressure (p (mbar)) using the Jena Climate Dataset (2009–2016). The dataset comprises a variety of meteorological parameters such as temperature, humidity, and wind speed. By leveraging various machine learning models and deep learning networks, this work aims to develop accurate and efficient predictive models for pressure estimation. Additionally, feature importance analysis provides insights into the relative significance of meteorological variables in influencing pressure. In this study, I will:

- Analyze important factors affecting the Air Pressure.
- Implement some Machine Learning models for estimating atmospheric pressure.
- Implement some Deep Learning models for estimating atmospheric pressure.
- Evaluate these implemented models.
- Suggest some improvements.

2 Features Chosen

2.1 Features Importance

Feature importance plays a crucial role in enhancing the accuracy and efficiency of machine learning models. By understanding and leveraging the importance of features in Random Forests, we can improve model performance and streamline training processes.

In machine learning, features (variables or attributes) represent measurable properties or characteristics of the observed phenomenon. These features serve as inputs to the model, and their selection can significantly impact the model's performance. Features are generally categorized into three types:

- **Numerical Features:** Quantitative data represented as numerical values.
- **Categorical Features:** Qualitative data indicating group membership.
- **Ordinal Features:** A subset of categorical features that have an inherent order or ranking.

Identifying important features provides several benefits:

- **Improved Model Accuracy:** Focusing on influential features can enhance prediction accuracy.
- **Reduced Training Time:** Streamlining the training process by prioritizing essential features saves time and computational resources.
- **Minimized Overfitting:** By emphasizing relevant features, models can avoid overfitting, ensuring better generalization to unseen data.

2.2 Random Forest

Random Forests, a robust ensemble learning method, creates multiple decision trees during training and aggregates their predictions to make final decisions. This technique is widely valued for its efficiency and interoperability.

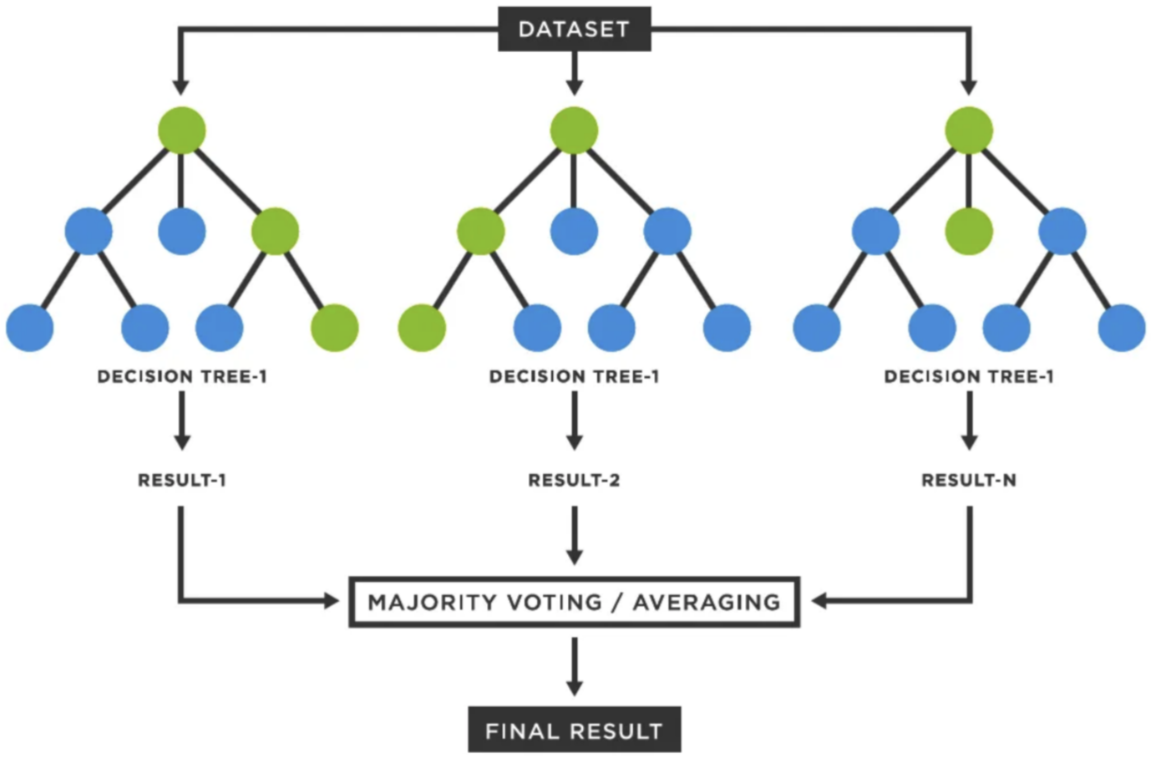


Figure 1: Sample model of Random Forest

Several approaches exist to measure feature importance in Random Forests:

- **Built-in Feature Importance:** Utilizes internal metrics like Gini importance or mean decrease in accuracy to evaluate the impact of features. These metrics quantify the reduction in impurity (randomness) within decision tree nodes when a feature is used for splitting.
- **Permutation Importance:** Assesses feature importance by permuting feature values and analyzing the resulting impact on model performance. This method measures the dependency of the model's accuracy on individual features.
- **SHAP (SHapley Additive exPlanations) Values:** Provides a comprehensive explanation of feature contributions to specific predictions, offering insights into feature importance across various data points.

By exploring and utilizing these techniques, we can gain a deeper understanding of feature importance, leading to better-performing models and efficient resource utilization.

2.3 Extract Features

The Random Forest Regressor is used to predict continuous numerical values and determine the importance of features contributing to these predictions. It evaluates feature importance based on how much each feature reduces impurity within the decision trees.

In Random Forest, each decision tree splits the data at various nodes using specific features. The reduction in impurity (e.g., variance reduction in regression tasks) caused by each feature is tracked. The `feature_importances_` attribute aggregates this information across all trees, assigning a score to each feature. Features that lead to significant impurity reductions are ranked as more important.

This approach provides a straightforward and efficient way to identify the most critical features for tasks such as predicting house prices, weather conditions, or other continuous outcomes. By understanding feature importance, the model's interpretability improves, and irrelevant or less significant features can be excluded, optimizing the regression process.

1. The dataset was split into training and testing sets, with a 50% split using `train_test_split`.
2. A Random Forest model with 100 trees (`n_estimators=100`) was trained on the training set.
3. Feature importances were extracted using the `feature_importances_` attribute of the trained model.
4. The results were stored in a pandas DataFrame, sorted in descending order of importance.
5. A bar plot was created to visualize the relative importance of each feature.

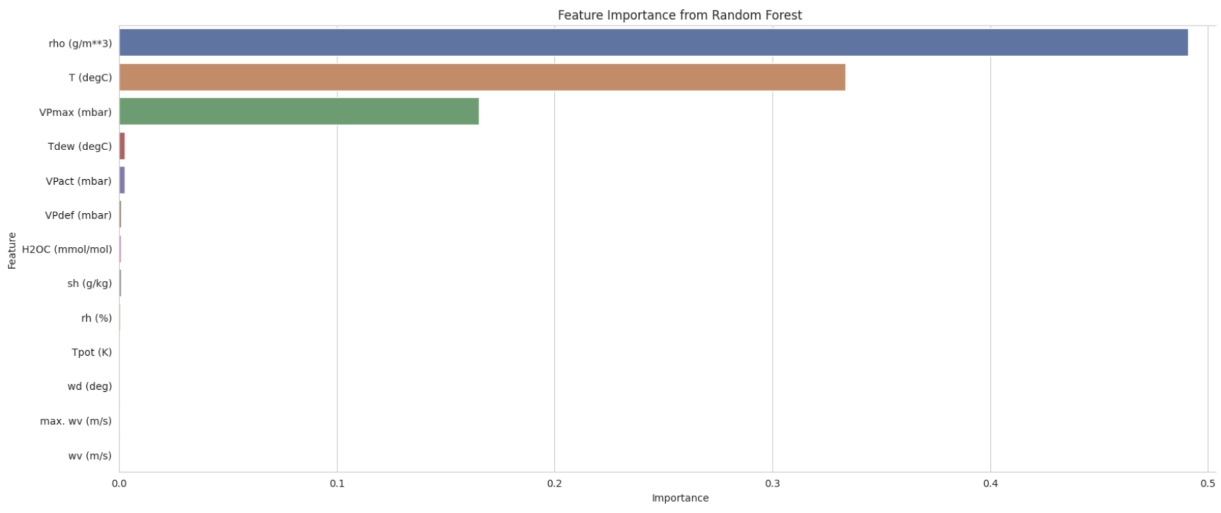


Figure 2: Random Forest Regressor to identify Important Features

The resulting plot provides insights into the relative importance of each feature in predicting the target variable Air Pressure `p (mbar)`.

It is obvious that having 3 specific features that significantly affect Air Pressure:

- Air Density - `rho (g/m**3)`.
- Temperature - `T (degC)`.
- Maximum Vapor Pressure - `VPmax (mbar)`.

3 Data Preprocessing

3.1 For Traditional LSTM Model

The raw Air Pressure data was normalized to improve training stability and performance. Normalization was performed using the mean and standard deviation of each feature, transforming the data into a standard normal distribution.

For the Traditional LSTM model, historical data was used to predict future values:

- **Input sequences (`X_sequences`):** Historical data was segmented into sliding windows of size 720, representing the past 120 hours (with a time step of 6 between samples).
- **Target sequences (`y_sequences`):** The next 72 time steps (equivalent to 12 hours) were used as the target for each input sequence.

The data preparation process included the following steps:

- Sliding windows of size 720 were created for the input features, with a step size of 6 to reduce redundancy.
- The target sequences were constructed as the next 72 time steps following each input sequence.

Input and target sequences were split into training (70%) and testing (30%) sets without shuffling, preserving the temporal order of the data.

3.2 For CNN-LSTM, XGB, and LGBM Model

7-to-7 Model

To reduce noise and maintain the trends and patterns of the signal, the raw Air Pressure data was first normalized and smoothed using a Savitzky-Golay filter. A polynomial order of 2 and a window size of 3 were chosen as the smoothing parameters. This preprocessing step effectively enhanced the model's ability to capture temporal dynamics.

Next, sequences were generated for supervised learning:

- **Input sequences (`X_sequences`):** The smoothed features were grouped into sliding windows of size 7. Each sequence contains 7 consecutive time steps, retaining temporal order for effective modeling.
- **Target sequences (`y_sequences`):** Corresponding target sequences were defined either as the next single time step or the next 7-time steps, depending on the requirements of the specific model.

The sequences were converted into `numpy` arrays and split into training (70%) and testing (30%) sets. Data shuffling was applied during splitting to ensure a balanced distribution between the two sets.

4 Deep Learning Models for Air Pressure Prediction problem

An artificial neural network (ANN), also known as a traditional neural network, is designed to handle simple tasks with a basic and straightforward structure. Loosely modeled after biological neural networks, it consists of multiple layers, each made up of interconnected nodes working together to perform specific functions.

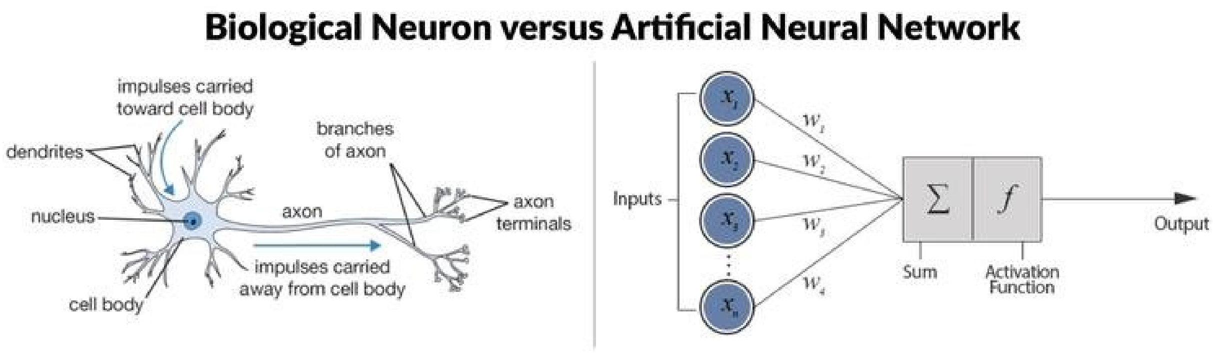


Figure 3: Analogy between biological and artificial neuron

These networks generally consist of an input layer, one or two hidden layers, and an output layer. While they can handle basic mathematical problems and computational tasks, such as logic gates with their truth tables, they are not well-suited for more advanced challenges like image processing, computer vision, or natural language processing.

To tackle these complex tasks, deep neural networks (DNNs) are used. These layers enhance the network's ability to understand complex problems and deliver effective solutions. With their increased number of layers (greater depth) compared to ANNs, DNNs add complexity while improving the network's ability to process inputs and generate optimal outputs.

4.1 LSTM

4.1.1 Concept

Long Short-Term Memory (LSTM), developed by Hochreiter and Schmidhuber, is an enhanced version of recurrent neural networks (RNNs).

Unlike traditional RNNs, which rely on a single hidden state passed through time and struggle with learning long-term dependencies, LSTMs address this issue by introducing a memory cell. This memory cell acts as a storage unit, allowing the network to retain information over extended periods.

Due to their ability to capture long-term dependencies in sequential data, LSTMs are highly effective for tasks such as language translation, speech recognition, and time series forecasting.

And the LSTM neural network can be characterized by the following equations: [?]

$$i_t = \text{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{1}$$

$$f_t = \text{sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2}$$

$$o_t = \text{sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{3}$$

$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{5}$$

$$h_t = \tanh(c_t) \odot o_t \tag{6}$$

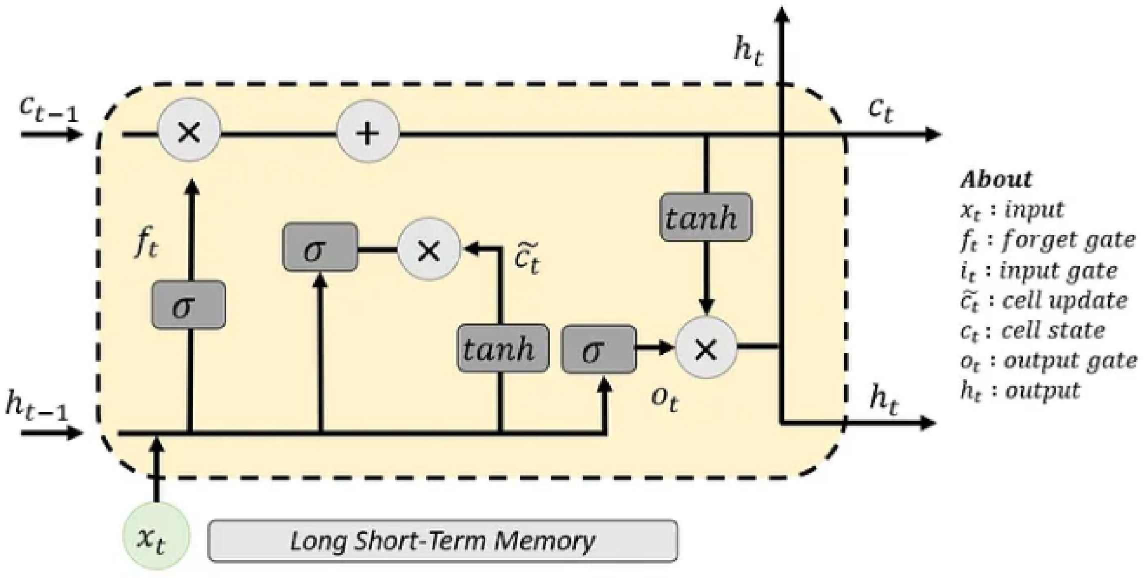


Figure 4: LSTM memory unit

Where x is the input vector of the layer, h is the output vector of the layer, $\odot$ is element-wise multiplication, W is the weight matrix, and b is the bias vector. i, f, o, g, c are respectively the input gate, forget gate, output gate, cell input, and cell activation. The subscript t means the current time step, and $t-1$ refers to the previous time step. As can be seen from the formulas, a hidden layer of LSTM contains eight different weight matrices.

The LSTM network has an input state that carries the current input information and a cell state that carries information from the previous time step, allowing the LSTM to make decisions based on the current input while considering the past sequence. This ability to capture and utilize both short-term and long-term dependencies is the key strength of LSTM in processing sequential data.

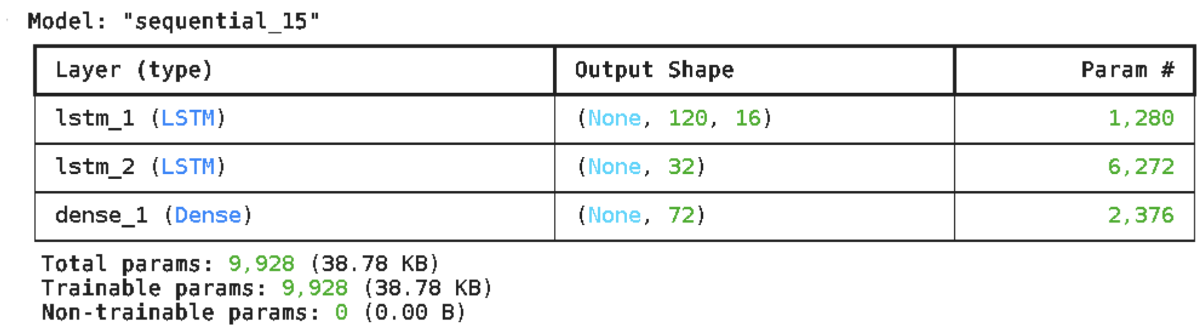
4.1.2 Implement Model and Output

The pressure is predicted based on the historical values of Air Pressure, Air Density, and Temperature. The Traditional LSTM model leverages its ability to capture long-term temporal dependencies for time-series prediction tasks.

Model Architecture and Training

The model architecture consists of two LSTM layers and a fully connected Dense layer:

- **Input Layer:** Accepts input sequences of shape `(batch_size, 720, 3)`, where 120 represents the historical time steps and 3 corresponds to the number of features (Air Pressure, Air Density, and Temperature).
- **LSTM Layers:**
  - The first LSTM layer has 16 units, with `return_sequences=True` to output the full sequence of hidden states for the next LSTM layer.
  - The second LSTM layer has 32 units, outputting the hidden state of the final state.
- **Dense Layer:** A fully connected layer with 72 units, corresponding to the predictions of 72 future time steps (next 12 hours, while each hour has 6 points of data).



```
Model: "sequential_15"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃     Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━┩
│ lstm_1 (LSTM)                    │ (None, 120, 16)        │       1,280 │
├──────────────────────────────────┼────────────────────────┼─────────────┤
│ lstm_2 (LSTM)                    │ (None, 32)             │       6,272 │
├──────────────────────────────────┼────────────────────────┼─────────────┤
│ dense_1 (Dense)                  │ (None, 72)             │       2,376 │
└──────────────────────────────────┴────────────────────────┴─────────────┘
 Total params: 9,928 (38.78 KB)
 Trainable params: 9,928 (38.78 KB)
 Non-trainable params: 0 (0.00 B)
```

Figure 5: LSTM model structure

The model was compiled with the following configuration:

- **Loss Function:** Mean Absolute Error (MAE), chosen for its robustness to outliers and its ability to minimize the average absolute difference between predicted and actual values.
- **Optimizer:** RMSprop with gradient clipping (`clipvalue=1.0`) to ensure stable training by preventing exploding gradients.
- **Evaluation Metric:** Root Mean Squared Error (RMSE), utilized to evaluate the model's performance on the validation set, providing a measure that is sensitive to larger prediction errors and indicative of overall accuracy.

The training process included:

- **Batch Size:** 256
- **Number of Epochs:** 50
- **Early Stopping:** Applied with a patience of 5 epochs to avoid overfitting, restoring the best weights.
- **Training Data:** Sliding window sequences of historical data (past 120 hours) and corresponding target sequences (next 12 hours).

Significance and Advantages

The proposed LSTM-based model offers the following benefits:

- **Long-Term Dependencies:** The LSTM architecture effectively captures relationships between historical and future values.
- **Accuracy:** The preprocessing steps (normalization and sliding windows) combined with the LSTM's architecture enable accurate future predictions.
- **Flexibility:** The model supports multi-step predictions, making it suitable for long-term forecasting.

Experimental Result

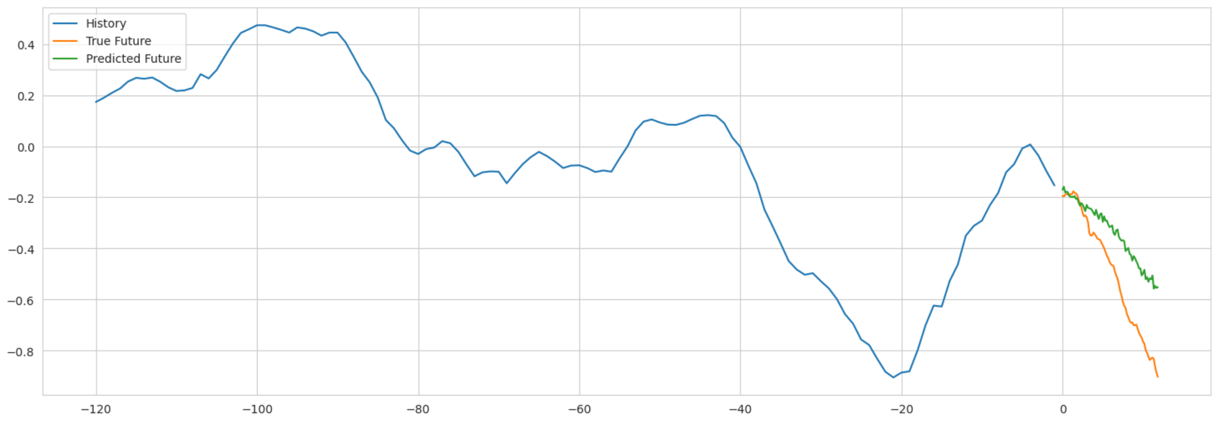


Figure 6: LSTM Prediction

The mean **RMSE** is 0.2239.

4.2 Combination between CNN and LSTM

4.2.1 Concept

A CNN-LSTM model combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks which results in a more intricate model. CNNs are used to extract spatial features, while LSTMs process temporal dependencies. This architecture is particularly effective for Air Pressure data that exhibits spatial patterns or localized features, making it suitable for multivariate time-series forecasting tasks.

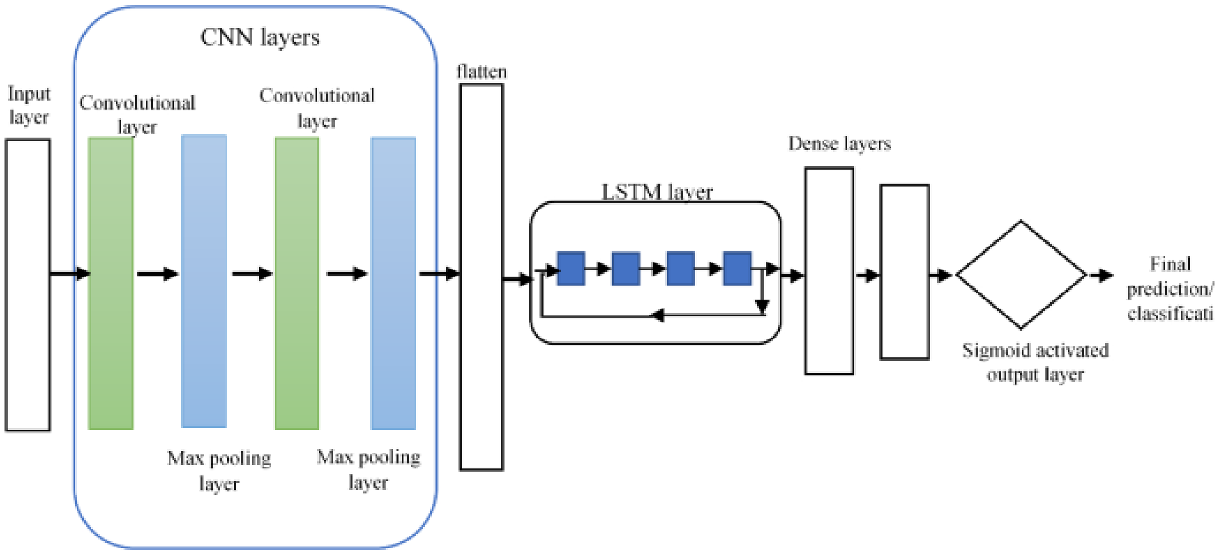


Figure 7: Sample of CNN-LSTM model

4.2.2 Implement Model and Output

The CNN-LSTM model predicts Air Pressure over the next 7-time steps (future horizon) using historical data of 7-time steps for Air Density, Temperature, and Maximum Vapor Pressure. The architecture efficiently captures spatial features via convolutional layers and temporal dependencies via LSTM layers.

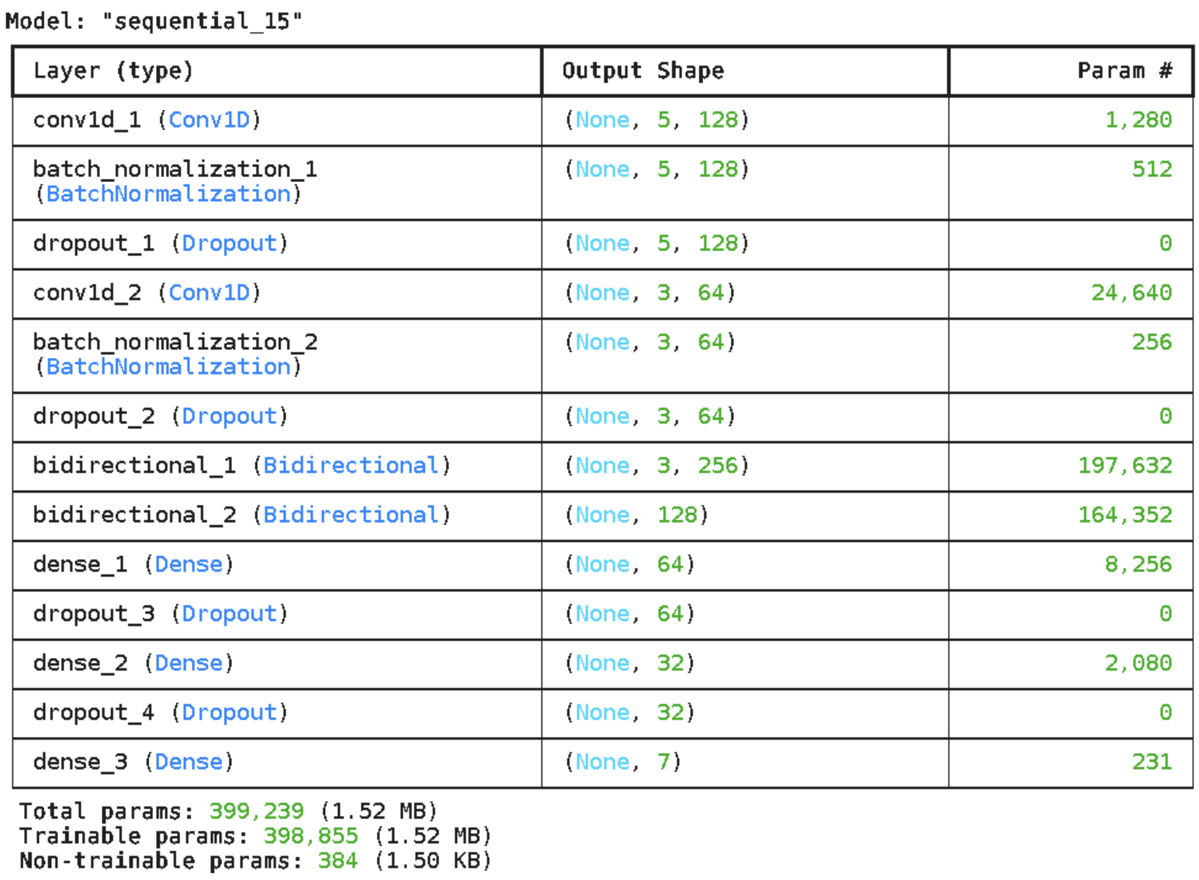
Model Architecture and Training

The CNN-LSTM model was implemented with the following architecture:

- **Input Layer:** Takes input sequences of shape `(batch_size, 7, 3)`, where 7 represents the sequence length and 3 corresponds to the number of features (Air Density, Temperature, and Maximum Vapor Pressure).
- **Convolutional Layers:**
  - **First Conv1D Layer:** 128 filters, kernel size of 3, and ReLU activation, followed by Batch Normalization and a Dropout layer with a rate of 0.2.
  - **Second Conv1D Layer:** 64 filters, kernel size of 3, and ReLU activation, followed by Batch Normalization and Dropout.
- **Bidirectional LSTM Layers:**
  - The first Bidirectional LSTM layer contains 128 units, returns sequences, and applies both dropout and recurrent dropout (0.2).

  - The second Bidirectional LSTM layer contains 64 units, with similar dropout configurations, but does not return sequences.

- **Fully Connected Layers:**
  - Dense layer with 64 units and ReLU activation, followed by Dropout (0.2).
  - Dense layer with 32 units and ReLU activation, followed by Dropout (0.2).
  - Output layer with 7 units for predicting the next 7-time steps.



Model: "sequential_15"

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 5, 128)	1,280
batch_normalization_1 (BatchNormalization)	(None, 5, 128)	512
dropout_1 (Dropout)	(None, 5, 128)	0
conv1d_2 (Conv1D)	(None, 3, 64)	24,640
batch_normalization_2 (BatchNormalization)	(None, 3, 64)	256
dropout_2 (Dropout)	(None, 3, 64)	0
bidirectional_1 (Bidirectional)	(None, 3, 256)	197,632
bidirectional_2 (Bidirectional)	(None, 128)	164,352
dense_1 (Dense)	(None, 64)	8,256
dropout_3 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2,080
dropout_4 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 7)	231

```
Total params: 399,239 (1.52 MB)
Trainable params: 398,855 (1.52 MB)
Non-trainable params: 384 (1.50 KB)
```

Figure 8: CNN-LSTM model structure

The model was compiled with:

- **Loss Function:** Mean Squared Error (MSE) minimizes the average squared difference between predicted and actual values, prioritizing larger errors for better accuracy on significant deviations.
- **Optimizer:** Adam optimizer with a learning rate of 0.001 ensures faster and more stable convergence, and is well-suited for handling non-stationary objectives and noisy data.
- **Metrics:** Mean Absolute Error (MAE), which provides an average measure of prediction error, and Root Mean Squared Error (RMSE), offering a more sensitive evaluation of larger errors.

The training process included:

- **Batch Size:** 64
- **Number of Epochs:** 18
- **Validation Split:** 10% of training data was used for validation.

- **Early Stopping:** Applied with patience of 5 epochs to prevent overfitting.
- **Learning Rate Scheduler:** `ReduceLROnPlateau` callback was used to dynamically reduce the learning rate by a factor of 0.2 if validation loss did not improve for 2 consecutive epochs.

Model Export and Deployment

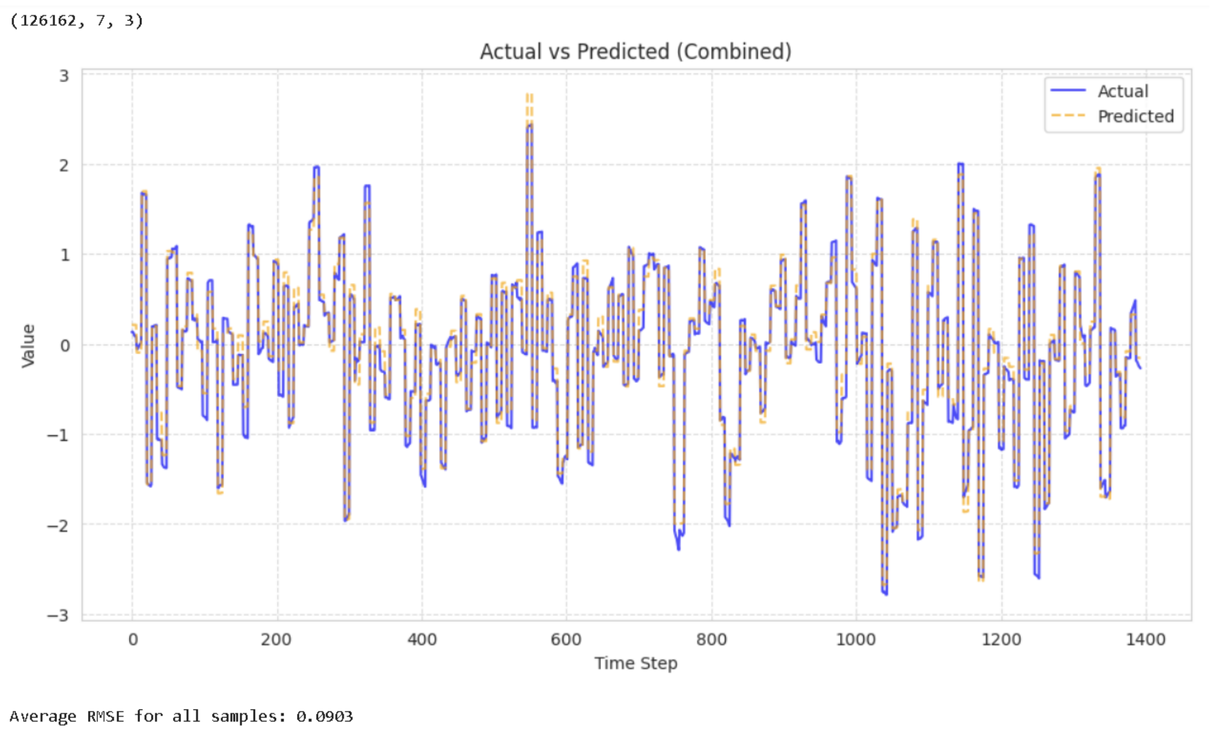
The trained CNN-LSTM model can be saved in HDF5 or TensorFlow Lite formats for deployment in edge devices or cloud-based applications. This enables real-time predictions for Air Pressure forecasting tasks in various environments.

Significance and Advantages

The CNN-LSTM model offers the following advantages:

- **Feature Extraction:** Conv1D layers efficiently capture spatial features and localized patterns in multivariate time-series data.
- **Temporal Dependencies:** LSTM layers enhance the model's ability to capture long-term temporal dependencies in sequential data.
- **Robustness:** Dropout and Batch Normalization reduce overfitting and improve model generalization.
- **Accuracy:** The combined CNN-LSTM architecture provides superior predictive performance for Air Pressure forecasting tasks compared to standalone models.
- **Scalability:** Modular architecture enables adaptation to other multivariate time-series prediction tasks with minimal changes.

Experimental Result



```
(126162, 7, 3)
```

```
Average RMSE for all samples: 0.0903
```

Figure 9: CNN-LSTM model prediction (7-to-7)

The mean **RMSE** is 0.0903.

5 Machine Learning Models for Air Pressure Prediction problem

5.1 XGB

5.1.1 Theory

XGBoost (Extreme Gradient Boosting) is a powerful ensemble learning technique that builds upon gradient boosting to create robust predictive models. It is highly efficient and scalable, using advanced optimization methods, parallel processing, and regularization techniques.

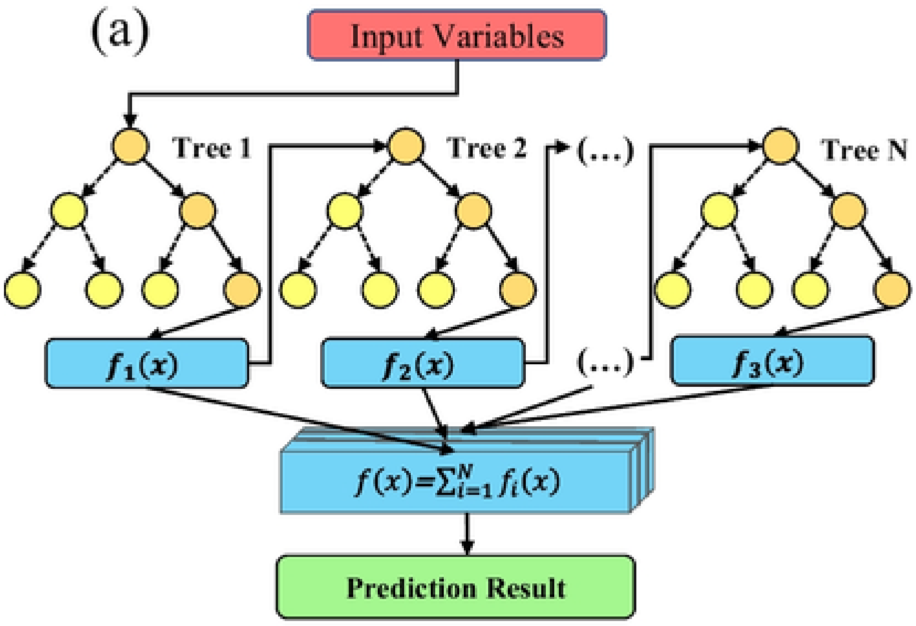


Figure 10: XGBoost Model

Key features of XGBoost include:

- Gradient boosting with improved speed and accuracy.
- Level-wise tree growth for controlled complexity.
- Regularization techniques to prevent overfitting.
- Support for sparse and missing data.

Gradient Boosting

At each iteration, XGBoost optimizes the loss function through:

- Computing first and second-order gradients for the loss function.
- Updating model predictions based on the gradients and Hessians.
- Adding trees iteratively to correct residuals from previous predictions.

Advantages include fast training, custom objectives support, and high-dimensional data handling.

Decision Tree Learning

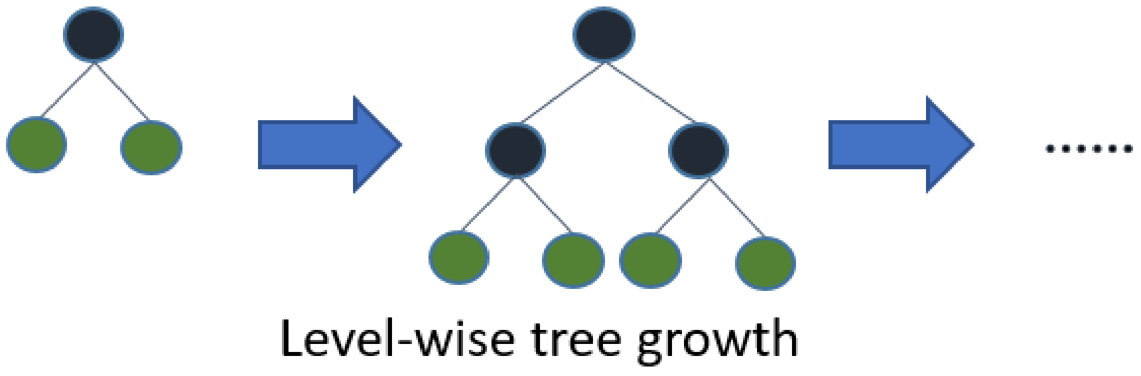


Figure 11: Level-wise growing

XGBoost uses level-wise tree growth:

- Expands the tree level by level, balancing complexity and computational efficiency.
- Splits nodes based on maximum gain in reducing the loss function.
- Handles missing data by assigning optimal splits based on the loss reduction.
- Supports categorical features and sparse matrices natively.

Regularization Techniques

XGBoost introduces regularization through:

- Leaf weights are subjected to regularization procedures to regulate model complexity.
- Tree-specific parameters such as `max_depth` and `min_child_weight`.
- Pruning of unnecessary branches to reduce overfitting.

5.1.2 Implement Model and Output

To predict Air Pressure targets based on sensor historical data, we employed the `XGBoost` framework with a regression setup (**XGBRegressor**). Instead of predicting classification labels, XGBRegressor predicts continuous values for regression tasks. The workflow included preprocessing input and target sequences, training individual models for each prediction step, and evaluating model performance using standard metrics.

For regression tasks, XGBRegressor supports the following **Loss Functions**:

- **Mean Squared Error (MSE):**

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  Used to minimize the mean squared error.

- **Mean Absolute Error (MAE):**

$$L = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Reduces the mean absolute error, less sensitive to outliers compared to MSE.

- **Custom Loss:** Combining MSE and MAE using custom gradients and Hessians for tailored objectives.

Model Architecture and Training

We used `XGBRegressor` with the following hyperparameters:

- **Objective:** Custom regression combining MSE and MAE.
- **Metric:** Root Mean Squared Error (RMSE).
- **Learning Rate:** 0.1
- **Number of Estimators:** 100
- **Maximum Depth:** 10
- **Subsampling Rate:** 0.8
- **Feature Subsampling Rate:** 0.8

For every step in the prediction horizon, a different model was trained, and each model used the input sequence to predict a certain output time step. The following procedure was used during training:

- Each model was evaluated on both the training and testing sets using RMSE and MAE metrics to ensure robust performance and accurate predictions across datasets.
- An iterative approach was employed to fine-tune the hyperparameters for each prediction step to achieve an optimal balance between accuracy and efficiency.
- The Modular approach allows each model to be deployed independently or integrated into larger forecasting pipelines.

This multi-step approach ensures that each model is tailored to predict its respective time step, effectively capturing temporal dependencies in the data while maintaining high accuracy and flexibility across the entire prediction horizon.

Model Export and Deployment

Trained models were serialized and exported using XGBoost's model-saving functionality. Each model for a prediction step was saved independently, ensuring modular deployment and scalability.

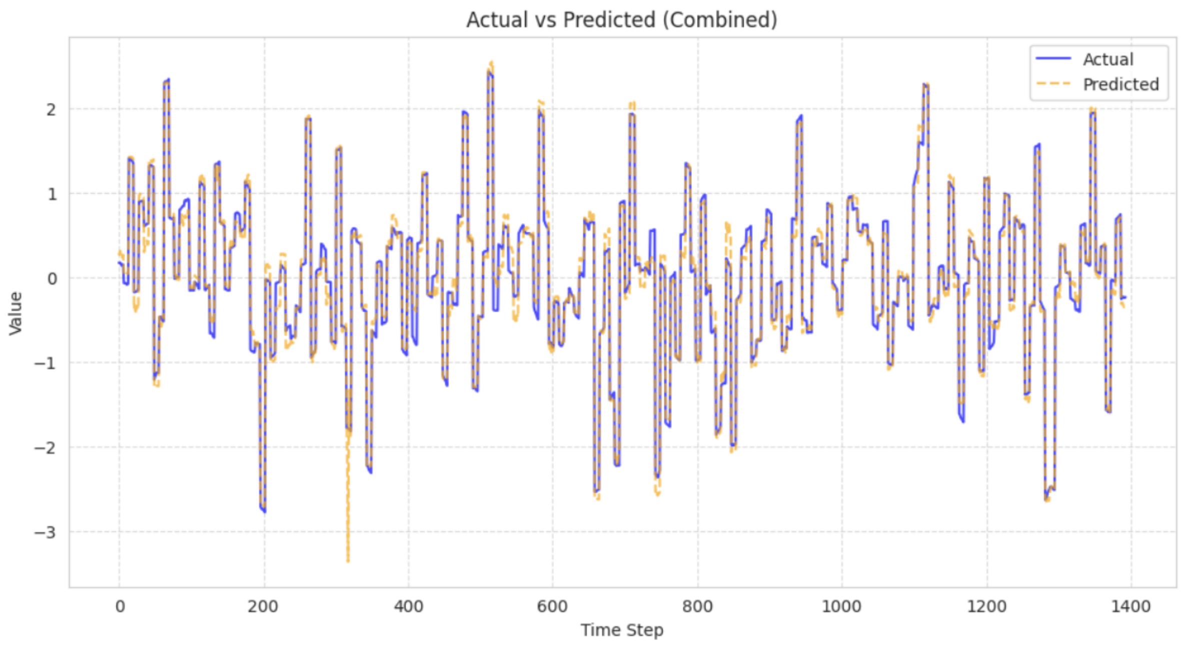
Significance and Advantages

The proposed model development pipeline offers the following benefits:

- **Efficiency:** XGBoost's gradient boosting approach ensures fast training and prediction times with optimized computation.
- **Scalability:** Separate models for each prediction step allow modular expansion and independent optimization for each time horizon.
- **Accuracy:** Smoothing the data reduces noise artifacts, supporting XGBoost focus on the underlying trends and patterns in the data.
- **Interpretability:** XGBoost models allow for the extraction of feature importances, which offer valuable information about the main variables affecting predictions.

Experimental Result

```
(126162, 21)
```



```
Average RMSE for all samples: 0.1006
```

Figure 12: Demo with XGB Model for 7-to-7 prediction

The mean **RMSE** is 0.1006.

5.2 Light GBM

5.2.1 Theory

LightGBM is a boosting algorithm designed to combine multiple weak decision trees into a robust predictive model. It is particularly effective due to its efficient gradient boosting, leaf-wise tree growth, and histogram-based approach.

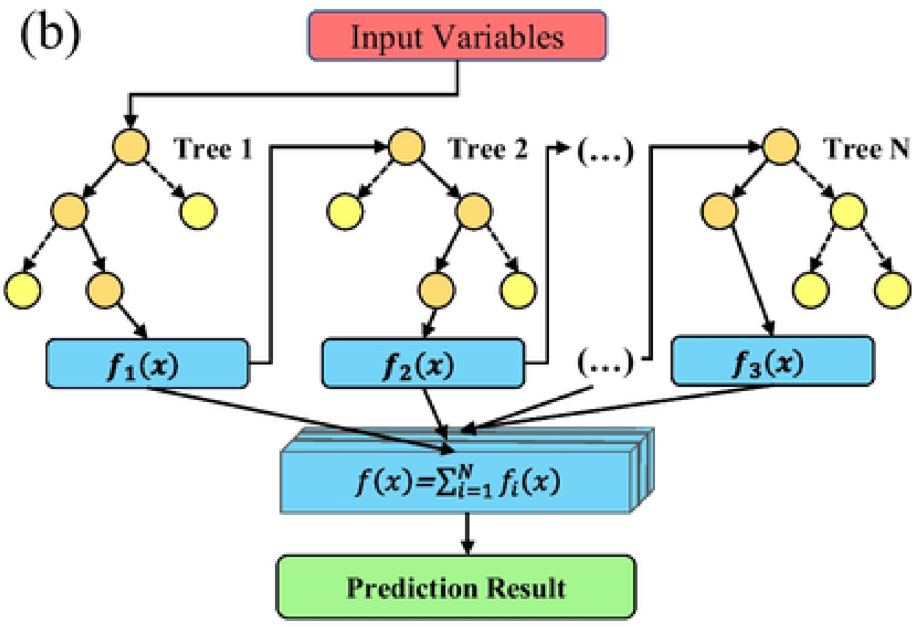


Figure 13: LightGBM Model

The algorithm starts with a single decision tree predicting the target variable. Subsequent trees are added iteratively to correct errors from previous iterations. Key steps include:

- Gradient boosting to minimize the loss function.
- Leaf-wise tree growth for efficient splitting.
- Histogram-based binning for computational efficiency.
- Regularization to prevent overfitting.

Gradient Boosting

At each iteration, gradient boosting:

- Computes the gradient of the loss function.
- Updates training weights using gradient information.
- Trains a new tree to correct prior errors.

Advantages include handling complex data, missing values, and nonlinear relationships.

Decision Tree Learning

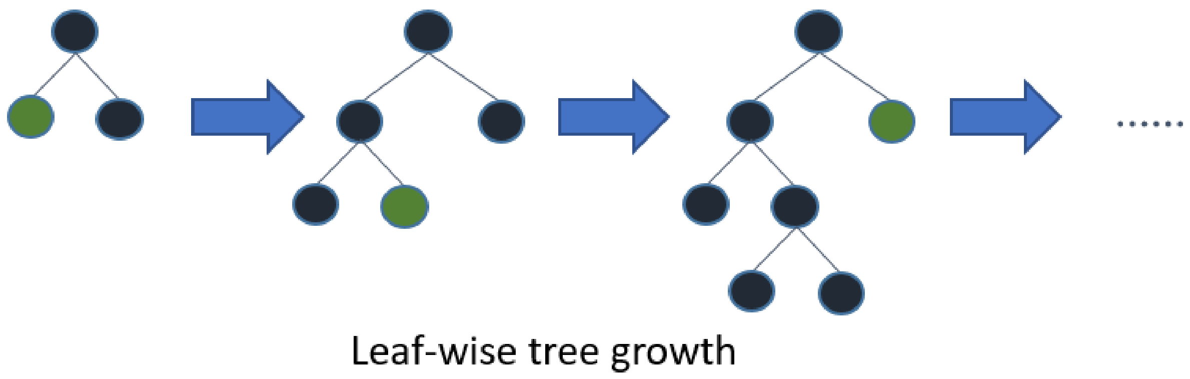


Figure 14: Leaf-wise growing

LightGBM uses decision tree learning to create weak learners:

- Splits data based on the most informative feature.
- Employs leaf-wise splitting for efficient feature selection.
- Handles missing data by assigning values to appropriate splits.
- Supports categorical splits for discrete data.

Histogram-Based Approach

LightGBM accelerates computation using histogram-based methods:

- Bins continuous features into discrete histograms.
- Computes gradients and Hessians for bins.
- Reduces memory usage and computation cost.

Key enhancements include:

- GOSS: Focuses on samples with large gradients, reducing data processed while maintaining accuracy.
- EFB: Groups sparse features into single bundles, reducing dimensionality and memory usage.

Leaf-Wise Tree Growth

LightGBM grows trees by expanding leaves with the highest gain:

- Focuses on splits that maximize loss reduction.
- Produces accurate and balanced trees.
- Uses regularization (e.g., max depth, minimum data per leaf) to avoid overfitting.

Regularization Techniques

Regularization methods include:

- Limiting tree depth to control complexity.
- Minimum data per leaf for statistical significance.
- Uses parameters to handle model regularization and constrain large coefficients.

5.2.2 Implement Model and Output

To predict Air Pressure targets based on sensor historical data, we employed the `LightGBM` framework with a tailored regression setup (**LGBMRegressor**). Instead of predicting classification labels, LGBMRegressor predicts continuous values for regression tasks. The workflow included preprocessing input and target sequences, training individual models for each prediction step, and evaluating model performance using standard metrics.

For regression tasks, LGBMRegressor uses different **Loss Functions** depending on the problem

- **Mean Squared Error (MSE):**
  $$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
  Used to minimize the mean squared error.
- **Mean Absolute Error (MAE):**
  $$L = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$
  Reduces the mean absolute error, less sensitive to outliers compared to MSE.
- **Huber Loss:** A combination of MAE and MSE to balance the impact of outliers.
- **Quantile Loss:** Suitable for quantile regression tasks, such as forecasting percentiles.

Model Architecture and Training

We utilized the `LGBMRegressor` from the LightGBM library, a gradient-boosting framework optimized for speed and scalability. The model was configured with the following hyperparameters:

- **Objective:** Regression
- **Metric:** Root Mean Squared Error (RMSE)
- **Learning Rate:** 0.05
- **Number of Estimators:** 200
- **Maximum Depth:** 5
- **Subsampling Rate:** 0.8
- **Feature Subsampling Rate:** 0.8
- **Maximum Leaves:** 31

Separate models were trained for each step in the prediction horizon, with each model predicting a specific output time step based on the input sequence. During training, the following process was applied:

- The model was evaluated on both the training and testing sets using the RMSE metric.
- An iterative approach was used to optimize performance for each prediction step.
- Models were saved in text format for reproducibility and further analysis.

This multi-step approach ensures that the model effectively captures the temporal dependencies in the data while maintaining high accuracy across all prediction steps.

Model Export and Deployment

Trained models were serialized and exported for deployment using LightGBM's native model-saving functionality. Each model corresponding to a prediction step was stored independently, enabling modular integration into downstream applications. This strategy ensures scalability and facilitates updates for individual steps without retraining the entire pipeline.

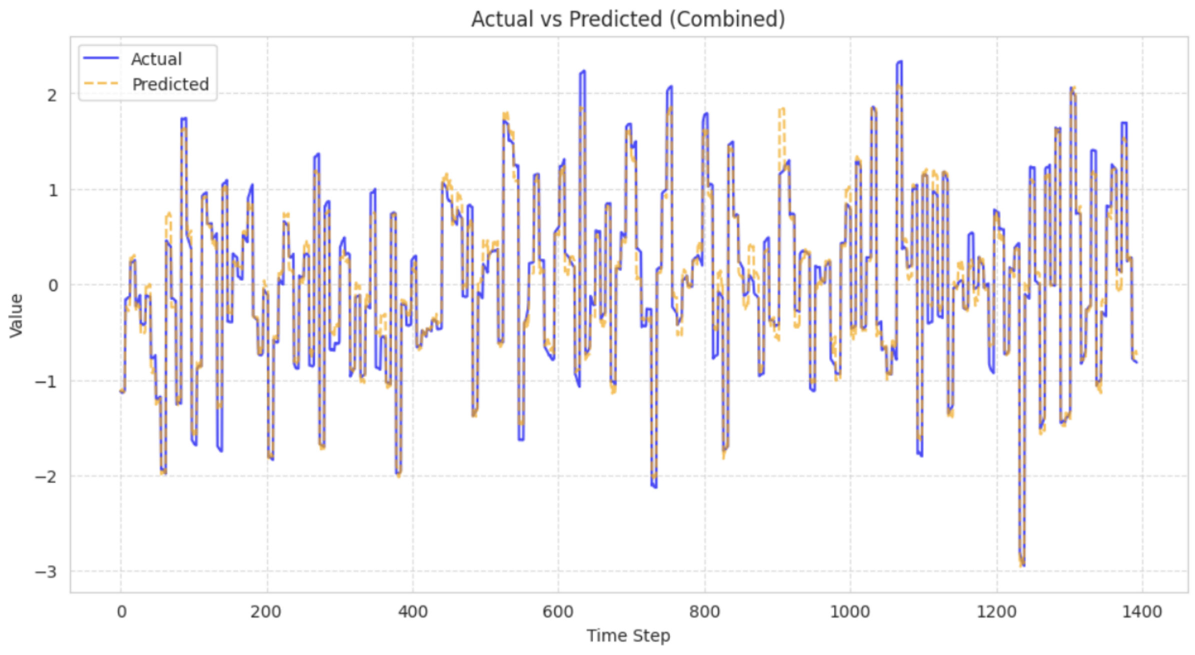
Significance and Advantages

The proposed model development pipeline offers the following benefits:

- **Efficiency:** LightGBM's gradient boosting approach provides fast training and prediction times.
- **Scalability:** Separate models for each prediction step allow modular expansion and adaptation.
- **Accuracy:** Smoothing of the data before training improves the robustness of the prediction by reducing noise-related artifacts.
- **Interpretability:** Feature importances can be extracted from LightGBM models, providing information on the factors that influence the predictions.

Experimental Result

```
(126162, 21)
```



```
Average RMSE for all samples: 0.1055
```

Figure 15: Demo with LGBM Model for 7-to-7 prediction

The mean **RMSE** is 0.1055.

6 Models Evaluation

The performance of the models was evaluated using the Root Mean Squared Error (RMSE). RMSE measures the average magnitude of the errors between predicted and actual values in a regression task.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where:

- y_i: The actual value of the target variable.

- $\hat{y}_i$: The predicted value from the model.

- n: The number of data points.

RMSE is a widely used metric owing to:

- It penalizes large errors more heavily due to squaring the differences.
- It has the same unit as the target variable, making it interpretable.
- A lower RMSE value indicates better model performance.

Each model was evaluated based on RMSE, and the results are summarized in the Table below. Lower RMSE values suggest that the model predictions are closer to the true values.

Model	**RMSE**
LSTM	0.2239
CNN-LSTM	0.0903
XGB	0.1006
LGBM	0.1055

On the Jena Climate dataset:

- The **LSTM model** achieved an RMSE of 0.2239. While capable of capturing long-term dependencies, it was slower and less efficient compared to other models. LSTM needs more data to train to get the better efficiency.
- The **CNN-LSTM** model had the lowest RMSE (0.0903), indicating that it effectively combined spatial and temporal feature extraction.
- Both **XGBoost (XGB)** and **LightGBM (LGBM)** performed well, with RMSE values of 0.1006 and 0.1055 respectively. These models were computationally efficient but required feature engineering to handle temporal dependencies.

7 Trade-Offs

- **LSTM (Long Short-Term Memory):** LSTM is an excellent choice for sequential data such as Air Pressure time series, as it can effectively capture long-term dependencies using its memory cells and gates. This makes it particularly suitable for tasks that involve temporal relationships. However, LSTM models are computationally expensive, requiring substantial training time, especially on large datasets. Additionally, they are prone to overfitting if not properly regularized and are generally slower than tree-based models like LightGBM or XGBoost due to their sequential computations.
- **Combination of LSTM and CNN:** The combination of LSTM and CNN enhances predictive capabilities by leveraging the strengths of both architectures. CNNs are adept at extracting spatial or local patterns, while LSTMs excel at capturing temporal dependencies. This hybrid approach is particularly effective when dealing with datasets that contain both spatial and sequential features. However, this model is computationally intensive and requires careful tuning of hyperparameters for both components. Its increased complexity also makes it less interpretable and more resource-demanding during training.
- **XGBoost:** XGBoost is a robust and efficient tree-based gradient boosting algorithm that achieves strong predictive performance through its regularization techniques to handle leaf weights, which help mitigate overfitting. It is well-suited for tabular data and handles missing values effectively. Compared to deep learning models like LSTM, XGBoost is faster to train and easier to interpret. However, it lacks the capability to directly capture temporal dependencies unless specific time-related features are engineered, which may limit its performance on purely sequential tasks.
- **LightGBM:** LightGBM focuses on speed and scalability, making it ideal for large datasets and distributed systems. Its leaf-wise tree growth strategy enables deeper trees, often resulting in higher accuracy on large datasets. However, this aggressive tree-growing approach can lead to overfitting on smaller datasets, necessitating careful tuning of parameters such as tree depth. Similar to XGBoost, LightGBM lacks the inherent ability to model temporal dependencies and relies on feature engineering to address sequential data tasks effectively.

8 Conclusion

In this study, we used four models: LSTM, CNN-LSTM, XGBoost, and LightGBM to predict Air Pressure using the Jena Climate Dataset. Each model had its own strengths and weaknesses, showing the trade-offs between deep learning and machine learning methods for time-series data.

The **LSTM** model was good at capturing patterns in sequential data because of its memory cells. However, it was slower to train and needed more computing power, which made it harder to use for large datasets. The **CNN-LSTM** combined the strengths of both CNN and LSTM. It could find spatial patterns with CNN and learn time-based patterns with LSTM. Although this model gave accurate results, it was complex and needed a lot of resources to train.

XGBoost and **LightGBM** were faster and easier to use. They worked well with large datasets and were more efficient than deep learning models. However, they needed extra work to include time-based features, as they cannot naturally handle sequential data. LightGBM was faster than XGBoost but could overfit if not tuned properly. Both models were good choices when speed and scalability were important.

In conclusion, the best model depends on the task. Deep learning models like LSTM and CNN-LSTM are better for complex time-series tasks but need more resources. XGBoost and LightGBM are faster and work well for large datasets but require feature engineering for sequential data. Choosing the right model depends on the specific needs and limitations of the problem.

You can have a look at our proposed models' implementation via **this link**.

References

[1] Prof. Nguyen Duc Dung. Machine Learning Lecture Materials. [HCMUT], [2024]. Includes:

- Chapter 02: Decision Tree
- Chapter 03: Deep Learning Networks
- Chapter 04: Regularization
- Chapter 06: Convolutional Networks

[2] Geeks for Geeks. Feature Importance with Random Forests. https://www.geeksforgeeks.org/feature-importance-with-random-forests/. Accessed: 12-2024.

[3] Geeks for Geeks. What is LSTM – Long Short Term Memory? https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/. Accessed: 12-2024.

[4] TensorFlow. Convolutional Neural Network (CNN). https://www.tensorflow.org/tutorials/images/cnn. Accessed: 12-2024.

[5] Mostafa Ibrahim. How to Use XGBoost and LGBM for Time Series Forecasting? https://365datascience.com/tutorials/python-tutorials/xgboost-lgbm/. Accessed: 01-2025.