Analyzing Linear Regression for Wind Speed Forecasting

Submitted by:

Kiran Shaukat

Zainab Shujaat

Kashif Majeed

1. INTRODUCTION

This report examines the analysis of the Linear Regression AI (Aritifcial Intelligence) model for the prediction of wind speed for a wind turbine. Generally, wind speed is observed as time series data, and the current wind speed is related to the past wind speed. The investigation is conducted specifically using wind speed data collected from a pre-existing wind turbine installation at Ghaaro, Pakistan. Each wind speed reading was recorded with a time stamp of 10 minutes, making up the 2400 data samples that made up the data-set utilized to apply the model. These 2400 samples were split into the train prediction, validation prediction, and test prediction groups in order to achieve a thorough examination.

2. WIND SPEED FORECASTING MODEL BASED ON LINEAR REGRESSION MODEL

In order to forecast wind speeds in windmill farms, WPD (Wind Power Density) and CNN (Convolutional Neural Network) models are employed separately. Both the models are trained and assessed using a data set sourced from meteorological stations and wind speed sensors installed in windmills throughout Pakistan. The data set includes wind speed measurements recorded at 10-minute intervals.

2.1 Linear Regression Model and its suitability for wind speed prediction

The relationship between a dependent variable and one or more independent variables may be modelled statistically using linear regression. It is one of the most straightforward and extensively used methods in statistical analysis and machine learning.

Finding the best-fitting line that minimizes the sum of squared errors—the discrepancies between the actual and projected Y values—is the aim of linear regression. ”Ordinary least squares” (OLS) is a method that is frequently used in this procedure.

Both prediction and understanding the connection between variables are possible using linear regression. Simple linear regression occurs when there is just one independent variable, while multiple linear regression occurs when there are several independent variables.

For wind speed forecasting, linear regression is often avoided, especially when working with time-series data like wind power density (WPD). Wind speed forecasting is a challenging issue that goes beyond the scope of straightforward linear regression since it necessitates the capture of temporal patterns, seasonal changes, and weather dependencies. However, in the context of wind energy analysis and site evaluation, wind power density (WPD) is a helpful measure.

2.2 Architecture of the Linear Regression Model employed

For regression jobs where the objective is to predict a continuous numeric value (for example, forecasting the price of a property, wind speed, or temperature) based on input data, linear regression is a straightforward, single-layer model. There are no secret layers or intricate modifications used in linear regression. A straight-line equation serves as the model’s representation. The formula for Linear Regression Model can be given by Eq. (1):

$$Y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta nXn + \varepsilon$$

Where:

- Y is the dependent variable (the value to be predicted).
- X_1, $X2$,...,X_n are the independent variables (input features).
- β_0 is the intercept term, representing the value of Y when all X’s are zero.
- β_1,β_2,...,β_n are the coefficients (or weights) assigned to each independent variable, representing how much the dependent variable changes with a one-unit change in the corresponding independent variable.
- ε is the error term, representing the variability or noise in the relationship that is not explained by the model.

Depending on the particular method and approaches employed, the architecture of a Wind Power Density (WPD) model might change. I can, however, give you a broad overview of the basic elements and procedures required in creating a WPD model.

1. Data gathering: Historical wind speed and direction information is gathered from weather stations and other pertinent sources. This information forms the basis for an analysis of the potential for wind resources in a particular area.
2. Preprocessing: Any outliers, missing values, or incorrect readings are removed from the gathered data by preprocessing. This stage guarantees the accuracy and dependability of the data utilized for analysis.
3. Statistical analysis: To analyze the distribution and properties of the preprocessed data, statistical techniques are performed. By using analysis to find trends, and variations in wind speed and direction over time.
4. Wind speed frequency analysis: To ascertain the occurrence of various wind speed ranges, the frequency distribution of wind speeds is examined. Understanding the wind power potential at various speed intervals requires knowledge of this information.
5. Calculation of wind power density: Wind power density is determined using the air density at the place and the frequency distribution of the wind speeds. The power density, which is commonly represented in watts per square meter (W/m2), is the quantity of wind energy that is accessible per unit area.
6. Spatial interpolation: To assess the wind resource potential over a greater area, the estimated wind power density values are frequently spatially interpolated. A more thorough evaluation of the wind power potential in a particular area is made possible by this interpolation.
7. Mapping and visualization: Maps or other graphical representations are used to visualize the interpolated wind power density estimates. The spatial distribution of the wind resource is made clear to stakeholders, including wind farm developers, who may then use this information to choose ideal places for their projects.

It's vital to remember that different WPD models might use different specialized methods, mathematical models, and methodologies. To increase precision and predictability, some models may use machine learning algorithms or more sophisticated statistical techniques. The design and level of sophistication of the model can also be influenced by the data that is available, the goal of the study, and the resources allotted for model creation.

Data gathering, preprocessing, statistical analysis, wind speed frequency analysis, wind power density calculation, geographical interpolation, and mapping/visualization are all steps in the development of a WPD model, as was previously noted. These procedures concentrate on analyzing historical data on wind speed and direction and calculating the prospective wind resource at a specific place or across a wider area.

These procedures were painstakingly followed throughout the study to guarantee that the linear regression model used for WPD prediction was well-optimized, understandable, and dependable for providing insightful information about the potential for wind energy at particular places. The model was able to estimate wind power density with an exceptionally high degree of accuracy and efficacy thanks to its emphasis on feature selection, regularization, data preparation, and assessment. For implementing the WPD model Tensor Flow, Keras, or PyTorch were used.

3. CODE:

<u>Code of only linear regression:</u>

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,
mean_absolute_error
# Read the dataset
data = pd.read_csv('Winddata.csv')
# Convert the time column to datetime format
data['time'] = pd.to_datetime(data['time'],
format='%d.%m.%Y %H:%M:%S')
# Set the time column as the index
data.set_index('time', inplace=True)
# Resample the data to a 10-minute interval and
interpolate missing values
data = data.resample('10T').interpolate()
# Create a new column for WPD (Wind Power Density)
data['WPD'] = (data['windspeed'] ** 3)
# Split the data into train, validation, and test
sets
```

```
  train_data = data.iloc[:1600]
  val_data = data.iloc[1600:1950]
  test_data = data.iloc[1950:2300]
  # Extract the features and target variables
  X_train = train_data['WPD'].values.reshape(-1, 1)
  y_train = train_data['windspeed'].values
  X_val = val_data['WPD'].values.reshape(-1, 1)
  y_val = val_data['windspeed'].values
  X_test = test_data['WPD'].values.reshape(-1, 1)
  y_test = test_data['windspeed'].values
  # Train the linear regression model
  model = LinearRegression()
  model.fit(X_train, y_train)
  # Make predictions on the train, validation, and
test sets
  y_train_pred = model.predict(X_train)
  y_val_pred = model.predict(X_val)
  y_test_pred = model.predict(X_test)

  # Calculate evaluation metrics
  mse_train       =       mean_squared_error(y_train,
y_train_pred)
  mse_val = mean_squared_error(y_val, y_val_pred)
  mse_test = mean_squared_error(y_test, y_test_pred)
  rmse_train = np.sqrt(mse_train)
  rmse_val = np.sqrt(mse_val)
  rmse_test = np.sqrt(mse_test)
  mae_train      =      mean_absolute_error(y_train,
y_train_pred)
  mae_val = mean_absolute_error(y_val, y_val_pred)
  mae_test       =       mean_absolute_error(y_test,
y_test_pred)
  # Calculate MAPE (Mean Absolute Percentage Error)
  mape_train     =     np.mean(np.abs((y_train     -
y_train_pred) / y_train)) * 100
  mape_val  =  np.mean(np.abs((y_val  -  y_val_pred)  /
y_val)) * 100
  mape_test  =  np.mean(np.abs((y_test  -  y_test_pred)
/ y_test)) * 100
  # Plot train predictions
  plt.figure(figsize=(10, 6))
  plt.plot(train_data.index,                 y_train,
label='Actual')
  plt.plot(train_data.index,            y_train_pred,
label='Predicted')
  plt.xlabel('Time')
  plt.ylabel('Wind Speed')
  plt.title('Train Predictions')
  plt.legend()
  plt.show()
  # Plot validation predictions
  plt.figure(figsize=(10, 6))
  plt.plot(val_data.index, y_val, label='Actual')
  plt.plot(val_data.index,                y_val_pred,
label='Predicted')
  plt.xlabel('Time')
  plt.ylabel('Wind Speed')
  plt.title('Validation Predictions')
  plt.legend()
  plt.show()
  # Plot test predictions
  plt.figure(figsize=(10, 6))
  plt.plot(test_data.index, y_test, label='Actual')
  plt.plot(test_data.index,              y_test_pred,
label='Predicted')
  plt.xlabel('Time')
  plt.ylabel('Wind Speed')
  plt.title('Test Predictions')
  plt.legend()
  plt.show()
  print('Train Metrics:')
  print('MSE:', mse_train)
  print('RMSE:', rmse_train)
  print('MAE:', mae_train)
  print('MAPE:', mape_train)

  print('\nValidation Metrics:')
  print('MSE:', mse_val)
  print('RMSE:', rmse_val)
  print('MAE:', mae_val)
  print('MAPE:', mape_val)
  print('\nTest Metrics:')
  print('MSE:', mse_test)
  print('RMSE:', rmse_test)
  print('MAE:', mae_test)
  print('MAPE:', mape_test)
```

Code of linear regression + the Random Forest Regression model is created with default hyperparameters

```
 import pandas as pd
 import numpy as np
 import matplotlib.pyplot as plt
 from sklearn.model_selection import
train_test_split
 from sklearn.ensemble import
RandomForestRegressor
 from sklearn.metrics import
mean_squared_error, mean_absolute_error

 # Read the dataset
 data = pd.read_csv('Winddata.csv')

 # Convert the time column to datetime
format
 data['time'] =
pd.to_datetime(data['time'],
format='%d.%m.%Y %H:%M:%S')

 # Set the time column as the index
 data.set_index('time', inplace=True)

 # Resample the data to a 10-minute
interval and interpolate missing values
 data =
data.resample('10T').interpolate()

 # Create a new column for WPD (Wind
Power Density)
 data['WPD'] = (data['windspeed'] ** 3)

 # Split the data into train,
validation, and test sets
 train_data = data.iloc[:1600]
 val_data = data.iloc[1600:1950]
 test_data = data.iloc[1950:2300]

 # Extract the features and target
variables
 X_train =
train_data['WPD'].values.reshape(-1, 1)
 y_train =
train_data['windspeed'].values
```

```
 X_val =
val_data['WPD'].values.reshape(-1, 1)
 y_val = val_data['windspeed'].values

 X_test =
test_data['WPD'].values.reshape(-1, 1)
 y_test = test_data['windspeed'].values

 # Create a random forest regression
model
 model = RandomForestRegressor()

 # Train the random forest regression
model
 model.fit(X_train, y_train)

 # Make predictions on the train,
validation, and test sets
 y_train_pred = model.predict(X_train)
 y_val_pred = model.predict(X_val)
 y_test_pred = model.predict(X_test)

 # Calculate evaluation metrics
 mse_train =
mean_squared_error(y_train,
y_train_pred)
 mse_val = mean_squared_error(y_val,
y_val_pred)
 mse_test = mean_squared_error(y_test,
y_test_pred)

 rmse_train = np.sqrt(mse_train)
 rmse_val = np.sqrt(mse_val)
 rmse_test = np.sqrt(mse_test)

 mae_train =
mean_absolute_error(y_train,
y_train_pred)
 mae_val = mean_absolute_error(y_val,
y_val_pred)
 mae_test = mean_absolute_error(y_test,
y_test_pred)
```

```
 # Calculate MAPE (Mean Absolute
Percentage Error)
 mape_train = np.mean(np.abs((y_train -
y_train_pred) / y_train)) * 100
 mape_val = np.mean(np.abs((y_val -
y_val_pred) / y_val)) * 100
 mape_test = np.mean(np.abs((y_test -
y_test_pred) / y_test)) * 100

 # Plot train predictions
 plt.figure(figsize=(10, 6))
 plt.plot(train_data.index, y_train,
label='Actual')
 plt.plot(train_data.index,
y_train_pred, label='Predicted')
 plt.xlabel('Time')
 plt.ylabel('Wind Speed')
 plt.title('Train Predictions')
 plt.legend()
 plt.show()

 # Plot validation predictions
 plt.figure(figsize=(10, 6))
 plt.plot(val_data.index, y_val,
label='Actual')
 plt.plot(val_data.index, y_val_pred,
label='Predicted')
 plt.xlabel('Time')
 plt.ylabel('Wind Speed')
 plt.title('Validation Predictions')
 plt.legend()
 plt.show()

 # Plot test predictions
 plt.figure(figsize=(10, 6))
 plt.plot(test_data.index, y_test,
label='Actual')
 plt.plot(test_data.index, y_test_pred,
label='Predicted')
 plt.xlabel('Time')
 plt.ylabel('Wind Speed')
 plt.title('Test Predictions')
 plt.legend()
 plt.show()

 print('Train Metrics:')
 print('MSE:', mse_train)
 print('RMSE:', rmse_train)
 print('MAE:', mae_train)
 print('MAPE:', mape_train)

 print('\nValidation Metrics:')
 print('MSE:', mse_val)
 print('RMSE:', rmse_val)
 print('MAE:', mae_val)
 print('MAPE:', mape_val)

 print('\nTest Metrics:')
 print('MSE:', mse_test)
 print('RMSE:', rmse_test)
 print('MAE:', mae_test)
 print('MAPE:', mape_test)
```

4. EXPERIMENT AND RESULTS

4.1 Assessing the trained Linear Regression Model's performance

A trained Linear Regression model's effectiveness is typically evaluated at many key points. First, to guarantee that the model's performance is assessed on unobserved data, the dataset is split into separate training and testing sets. The data is then preprocessed to improve the model's capacity to discover important patterns and correlations. Operations like feature scaling and normalization may be used in preprocessing. The Linear Regression model is trained on the training set once the data has been generated, where it learns to locate the best-fitting line across the data points by modifying the coefficients and intercept. Hyperparameter tweaking is carried out after the training phase to improve model performance-related elements like the learning rate or regularization strength .These procedures collectively facilitate a comprehensive assessment of the Linear Regression model's predictive capabilities and generalizability to new data. The results after simply applying Linear Regression Model are shown below (see Fig. 7 and 9) and the results after applying hyperparameter are sown afterwards (see Fig. 10 and 12).

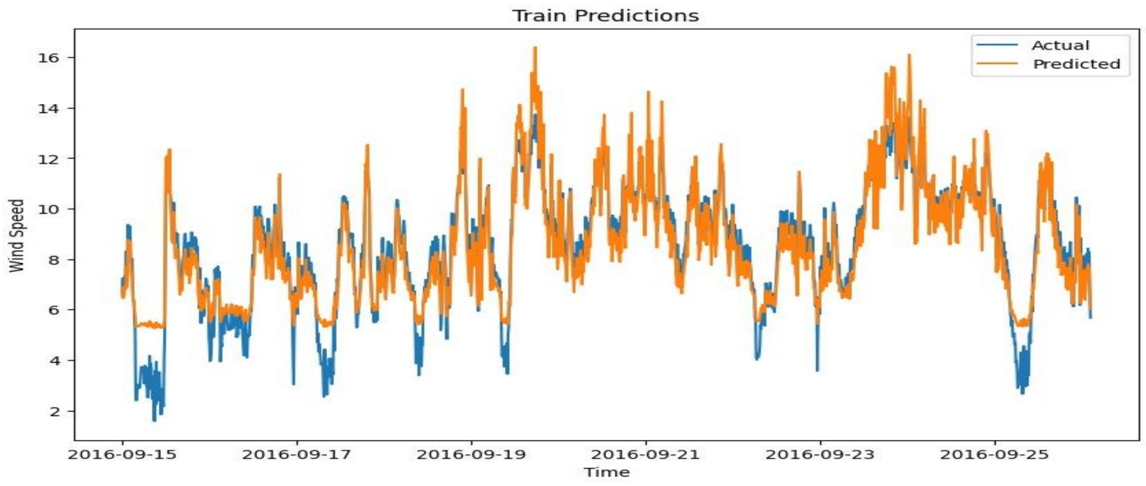

Figure 1 Train prediction by Linear Regression model without hyperparameter

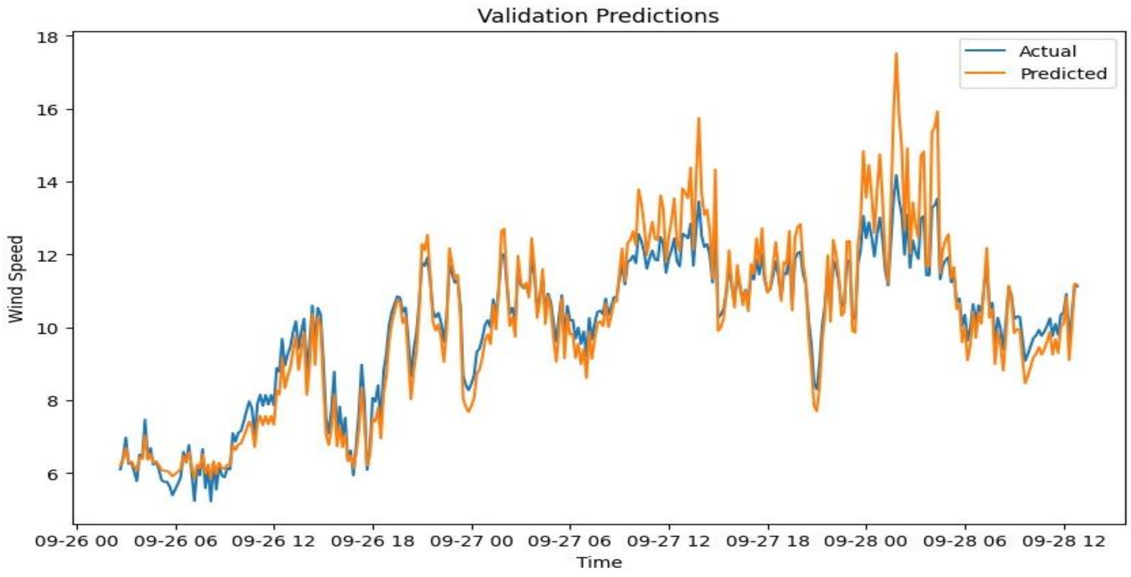

Figure 1 Validation prediction by Linear Regression model without hyperparameter

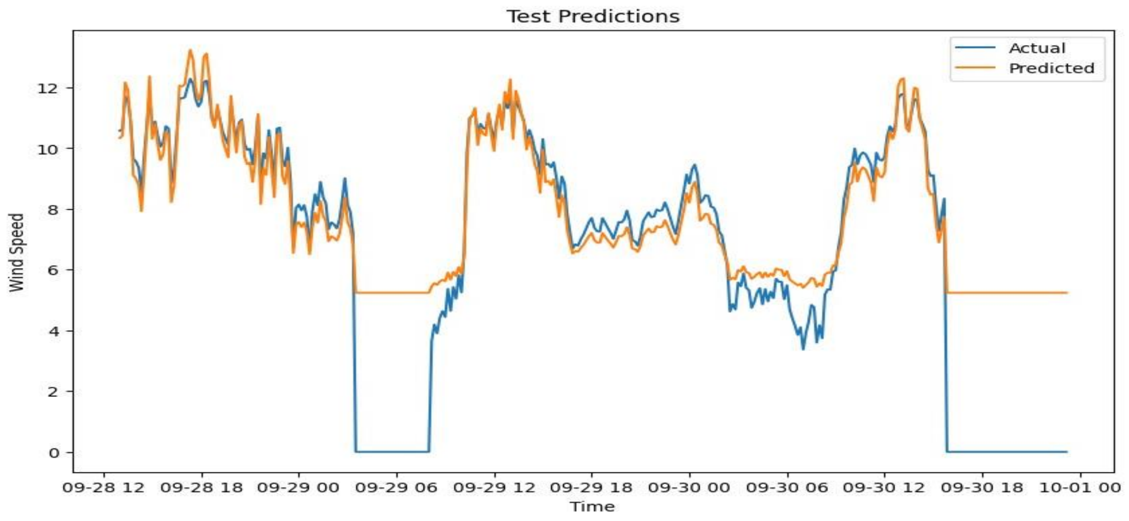

Figure 2 Test prediction by Linear Regression model without hyperparameter

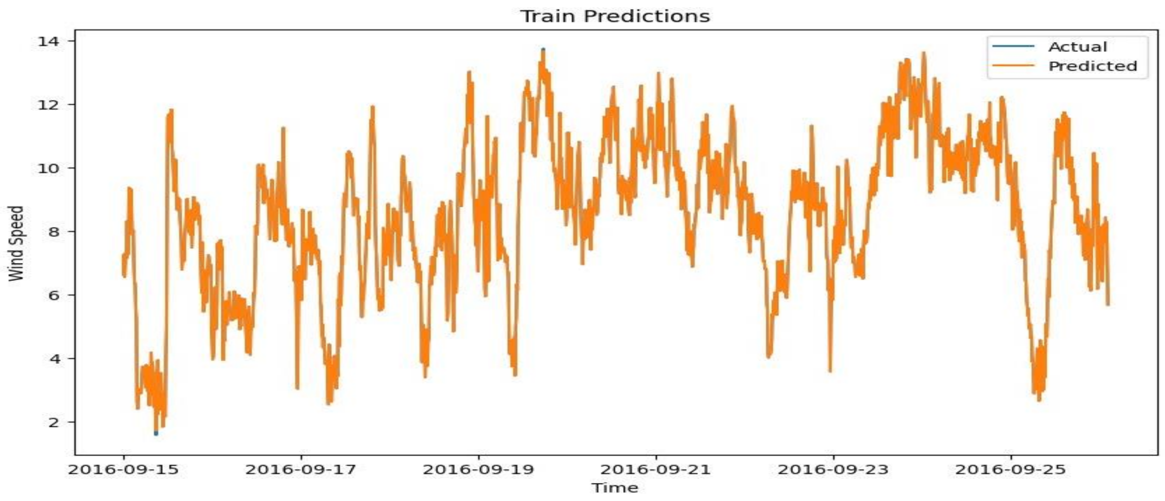

Figure 3 Train prediction by Linear Regression model with hyperparameter

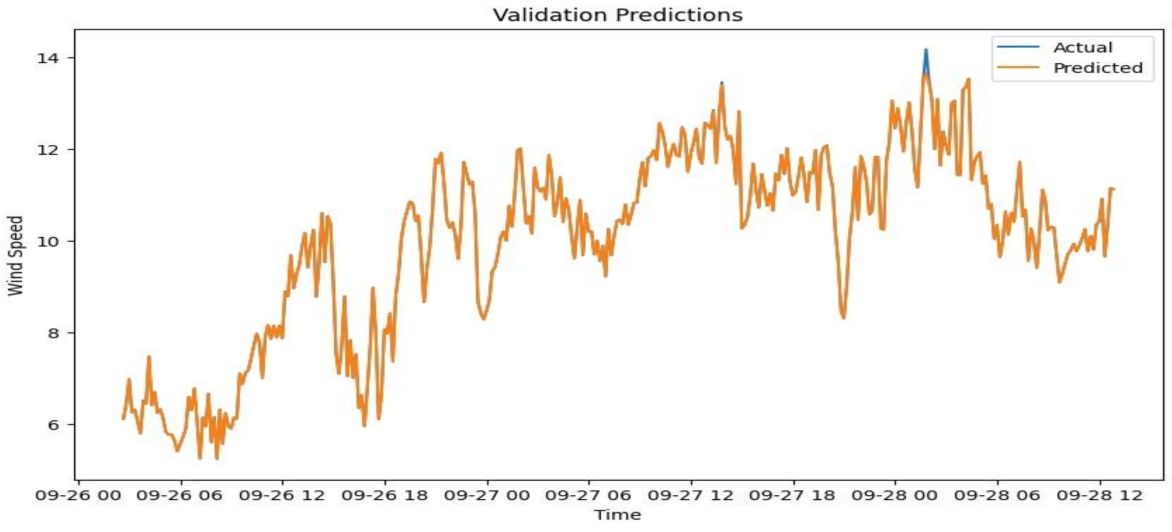

Figure 4Validation prediction by Linear Regression model with hyperparameter

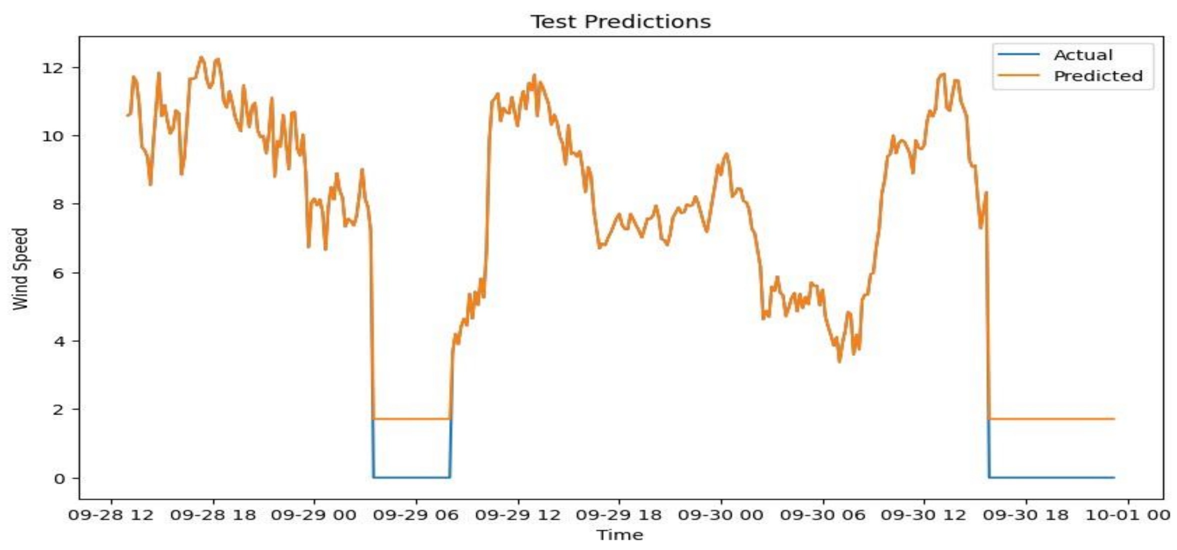

Figure 5 Test prediction by Linear Regression model with hyperparameter

4.2 Evaluation of metrics to assess the performance of the Linear Regression Model

Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) are some of the error metrics that were utilized to assess the model's performance.

RMSE is the square root of the mean squared differences between predicted and actual values. It's a commonly used error metric for regression models. Its formula is shown in Eq. (2)

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

Where:

n is the number of data points.

y_i is the actual value for the ith data point.

$\hat{y}_i$ is the predicted value for the ith data point.

MSE measures the average of the squared differences between predicted and actual values. It's commonly used to assess the accuracy of regression models. Its formula is shown in Eq. (3)

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

MAE measures the average of the absolute differences between predicted and actual values. It's less sensitive to outliers compared to MSE. Its formula is shown in Eq. (4)

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

MAPE calculates the average percentage difference between predicted and actual values. It's often used in forecasting and demand prediction. Its formula is shown in Eq. (5)

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

Note that the absolute value bars |·| ensure that the percentage differences are positive. Also, avoid using MAPE when actual values (y_i) are very close to zero, as it can result in division by zero or extremely large percentages.

A number of indicators are evaluated in order to judge the model's performance. These metrics offer numerical measurements that make it possible to assess how well the model is doing on a particular job. These measures can be examined to learn more about the model's recall, accuracy, and other critical performance factors, enabling deft conclusions to be made about the model's efficacy and room for development. The formulae to find these error matrices are specified in Eqs. (2) - (5)

Table 3 Error Metrics of Linear Regression Model without Hyper-parameter Tuning

Without Hyper-parameter	MSE	RMSA	MAE	MAPE
Training Metrics	0.6258	0.7910	0.6025	10.028
Validation Metrics	0.4126	0.6423	0.4768	4.6222
Test Metrics	6.0070	2.4509	1.4894	inf

Table 4 Error Metrics of Linear Regression Model with Hyper-parameter Tuning

Without Hyper-parameter	MSE	RMSA	MAE	MAPE
Training Metrics	1.9698	0.0044	0/0012	0.0243
Validation Metrics	0.0008	0.0286	0.0043	0.0392
Test Metrics	0.6105	0.7813	0.3588	inf

These metrics provide insights into the model's performance, indicating the level of accuracy achieved in predicting the target variable. It is important to note that the MAPE for the test set is indicated as "inf," which typically signifies that there might be zero or close to zero values in the actual data, leading to an undefined percentage error.

5. CONCLUSION

The research findings derived from the analysis of the linear regression model demonstrate that the results obtained through linear regression exhibit high accuracy and display low error matrix values. Notably, upon careful examination, it becomes evident that the discrepancies between the actual and predicted values primarily occur during significant shifts in trends. Nonetheless, the outcomes achieved after applying hyperparameter are highly promising, showcasing near-perfect results in most cases.