Feature-Based Ensemble Learning for PV Forecasting

Debanuj Roy[1,2*]

[1*]Mechanical Engineering , National Institute of Technology, Durgapur, Durgapur, West Bengal, India.

Corresponding author(s). E-mail(s): dr.23me8179@nitdgp.ac.in;

Keywords: keyword1, Keyword2, Keyword3, Keyword4

1 Introduction

Time Series forecasting is a fundamental problem in data science with applications in fields such as finance, climatology, and resource management. Accurate predictions enable better decision- making, optimizing resources, and reducing uncertainty in planning processes. Various approaches exist for time series forecasting, including statistical models like SARIMA, deep learning models such as LSTM, and multivariate models like VAR. This study examines the performance of these four models using a dataset with weekly observations. SARIMA is applied to capture linear trends and seasonality, utilizing pmdarima's ndiffs() function to determine the optimal differencing order. LSTM, a deep learning- based approach, is trained on sequences of past values to predict future points. The VAR model is implemented with two features, incorporating an additional variable (day length) to improve predictive accuracy and lastly, the Transformer which could is also trained on past sequences and performs exceptionally well on sequential data due to the self-attention mechanism which lets each timestep "attend" to every other token,learning which parts of the sequence are most relevant to predicting the output at any step.Each model's inference is evaluated on test data to assess its strengths and limitations in capturing temporal dependencies.Lastly;a simple averaging ensemble model,a CNN Model and a MLP Model is used to give a final consolidated result.

2 Dataset Used

The dataset used in this study is sourced from the Kaggle repository: Solar Power Generation Data. It consists of time-indexed records specifically focusing on solar power generation metrics. The dataset contains key variables such as the amount of solar energy generated, along with additional features like day length, temperature, and atmospheric conditions that may infiuence power output. The dataset contained a large amount of observations which made training the models difficult therefore we have constructed a new dataset from the existing ones by taking average observations over a week.We have also applied MinMaxScaler from Scikit-learn library to scale the values to improve performance by preventing exploding gradient when using ReLU activation.

3 Models Used

3.1 Utilizing Time Series Concepts

3.1.1 SARIMA (Seasonal Autoregressive Integrated Moving Average)

3.1.2 VAR (Vector Autoregression)

3.2 Utilizing Deep Learning Neural Networks

3.2.1 LSTM (Long Short-Term Memory)

3.2.2 Transformer

3.3 Ensembling Models

3.3.1 MLP (Multi-Layer Perceptron)

3.3.2 CNN (Convolutional Neural Network)

3.3.3 Simple Average

4 Abstract

Accurate long-term forecasting of regional photovoltaic (PV) generation is crucial for reliable grid integration and energy man- agement. This paper proposes a novel feature-selective ensemble learning framework for long-term regional PV generation forecasting. The method integrates diverse predictive models, including statistical and machine learning techniques, to capture the complex and nonlinear relationships inherent in PV generation.Subsequently, an ensemble learning strategy is employed, where the individual model predictions are combined using an averaging approach. The efficacy of the proposed method is evaluated using real-world regional PV generation data, demonstrating significant improvements in forecasting accuracy compared to individual models and traditional ensemble methods. The results highlight the potential of feature-selective ensemble learning for enhancing the reliability of long-term regional PV generation forecasting, contributing to improved grid stability and efficient renewable energy integration.

5 Overview Of The Models

5.1 SARIMA

The Seasonal Autoregressive Integrated Moving Average (SARIMA) model, a cornerstone of time series forecasting, models linear rela- tionships and trends through

Autoregression (AR), Differencing (I), and Moving Average (MA) components. A key SARIMA require- ment, stationarity (constant mean, variance, and absence of trend or seasonality), was achieved by differencing. The optimal differenc- ing order was determined using pmdarima's ndiffs() function and validated with ADF and KPSS tests. Leveraging the dataset's weekly seasonality, the model was implemented, incorporating a 52-period differencing term. AR and MA orders(p,q) were derived from PACF and ACF plots. Model parameters were optimized to minimize fore- cast errors, and performance was objectively evaluated on a test set using standard error metrics.The mathematical representation of the SARIMA Model:-

$$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \cdots - \theta_q \epsilon_{t-q}$$

where y_{t-n} are the values of the time series at a lag of n from 0 to p, and ϵ_{t-m} are the errors at a lag of m from 0 to q.

5.2 Vector Autoregression (VAR)

The Vector Autoregressive (VAR) model is a multivariate time series forecasting technique that captures the linear correlations among multiple variables. Unlike univariate models like SARIMA, VAR considers multiple time series simultaneously, making it effective for forecasting scenarios with multiple infiuencing factors. In this study, the VAR model was implemented using two features: solar power generation and day length, to improve predictive accuracy. The dataset was first checked for stationarity using the Augmented Dickey-Fuller (ADF) test, and differencing was applied where necessary. The optimal lag order was determined using the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). The model was trained on sequences of past 25 values, using a rolling forecast approach for evaluation. The mathematical formula for the VAR model is given by:-

$$\mathbf{y}_t = \mathbf{c} + \Phi_1 \mathbf{y}_{t-1} + \Phi_2 \mathbf{y}_{t-2} + \cdots + \Phi_p \mathbf{y}_{t-p} + \boldsymbol{\varepsilon}_t \tag{2}$$

where $\mathbf{y}_t$ consists of multiple correlated feature vectors at lag x, p is the lag order, Φ_i is the coefficient matrix for lag i, and $\boldsymbol{\varepsilon}_t$ is the white noise.

5.3. LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) specifically designed to capture long-term dependencies in sequential data. Unlike traditional RNNs, which suffer from vanishing gradient problems, LSTMs use memory cells and gating mechanisms to retain important information over extended sequences.

In this study, an LSTM-based model was constructed to predict solar power generation. The model was trained on sequences of the past 52 values to predict the next time step. The network architecture consists of multiple LSTM layers followed by dense layers, optimized using backpropagation. The Adam optimizer and Mean Squared Error (MSE) loss function were used to fine-tune the model for optimal performance.

An LSTM cell consists of three primary gates:

Forget Gate: The forget gate determines what information should be discarded from the cell state. It takes the previous hidden state and the current input, then applies a sigmoid activation function to generate a forget factor between 0 and 1:

$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right)$$

where:

- f_t is the forget gate vector,
- W_f and b_f are the weight matrix and bias,
- σ represents the sigmoid activation function.

If f_t is close to 1, the information is retained; if close to 0, the information is forgotten.

Input Gate: The input gate decides which new information should be added to the cell state. It consists of two parts: a sigmoid gate to determine the importance of new input and a tanh function to generate a candidate cell state update:

$$i_t = \sigma\left(W_i[h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh\left(W_C[h_{t-1}, x_t] + b_C\right)$$

where:

- i_t is the input gate activation vector,
- $\tilde{C}_t$ represents the candidate cell state,
- W_i, W_C are the weight matrices,
- b_i, b_C are the biases.

The new cell state is then updated as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate: The output gate determines the next hidden state based on the updated cell state. The sigmoid function decides which parts of the cell state should be exposed as output, followed by a tanh activation:

$$o_t = \sigma\left(W_o[h_{t-1}, x_t] + b_o\right)$$
$$h_t = o_t * \tanh(C_t)$$

where:

- o_t is the output gate vector,
- h_t is the hidden state output,
- W_o and b_o are the output gate parameters.

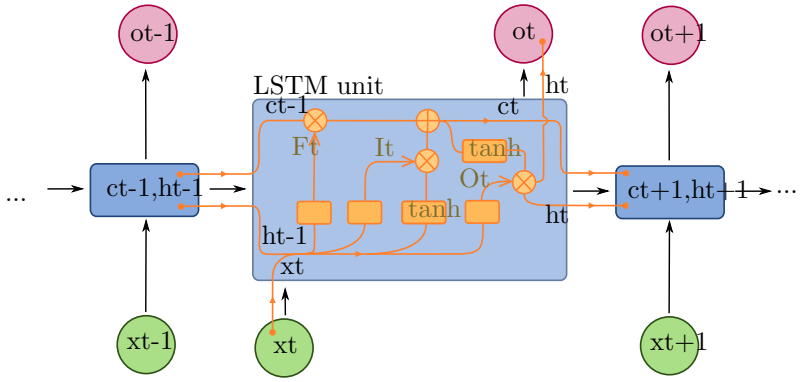

Fig. 1 Structure of an LSTM cell (source: Wikimedia Commons)

5.3 5.4 Transformer

Because it can capture long-range dependencies without depending on recurrence, the Transformer architecture—which was first proposed for natural language processing—has shown promise for time-series forecasting. Transformers improve computational efficiency and learning performance by using self-attention mechanisms to analyze relationships between all time steps in parallel, in contrast to RNNs or LSTMs that process data sequentially.

52 weekly solar energy readings were used as input in this study's implementation of the Transformer model. An input embedding layer and several encoder

layers, each made up of feed-forward and multi-head self-attention sublayers, make up the architecture. To preserve the data's temporal structure, positional encoding was introduced.

The attention mechanism can be summarized as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where Q (query), K (key), and V (value) are learned projections of the input, and d_k is the dimension of the key vectors.

The next anticipated time step was produced by passing the altered sequence through dense layers after it had been processed through encoder blocks. The Mean Squared Error (MSE) loss function and Adam optimizer were used to train the model.

Transformers are appropriate for complex, high-dimensional time-series forecasting tasks like PV energy prediction because they can parallelize training and provide robustness against long-term dependencies.

5.5. Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a class of feedforward neural networks consisting of multiple layers of neurons. It typically comprises an input layer, one or more hidden layers, and an output layer.

The network is trained using backpropagation and optimized using gradient descent techniques.

The mathematical representation of a neuron in an MLP is:

$$y = f(Wx + b)$$

where:

- x is the input vector,
- W is the weight matrix,
- b is the bias term,
- f is the activation function (e.g., ReLU, sigmoid, or tanh),
- y is the output.

MLPs are widely used in regression and classification tasks, making them suitable for time-series forecasting when properly structured.

5.6. Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a deep learning model designed for feature extraction and pattern recognition. CNNs consist of convolutional layers, pooling layers, and fully connected layers, making them effective in both image processing and sequential data modeling. In time-series forecasting, CNNs capture spatial and temporal dependencies by applying kernel filters that extract hierarchical features from the input sequence.

Key components used in the CNN model:

- **Conv2D (Two-Dimensional Convolutional Layers)**: Applies kernel filters to extract useful spatial features, producing 2D feature maps.
- **MaxPooling Layers**: Reduce the spatial dimensions of the feature maps while preserving important information, lowering computational cost.

5.7. Simple Average

The simple average method involves computing the mean of the predictions generated by each base model (LSTM, SARIMA,VAR and Transformer). This ensemble technique helps to reduce individual model bias and improve overall forecast robustness by leveraging the strengths of each meta-learner.

6. Methodology

This section outlines the methodology used for time-series forecasting using the above machine learning models.The key stages include data preprocessing,model implementation,ensembling and evaluation.

6.1 Data Preprocessing

The dataset was sourced from a CSV file and prepared using the following steps:

- Imported Python libraries: `pandas`, `numpy`, `matplotlib`, `seaborn`, and `scikit-learn`.
- Loaded the dataset and filtered the required number of observations.
- Converted timestamps into structured datetime format.
- Aggregated data to weekly averages to reduce size and training time.
- Split the dataset into training and testing sets.

6.2 Implementation of Models

Multiple machine learning models were implemented for forecasting. The implementation of each is described below:

6.2.1 SARIMA:

The SARIMA model was configured as follows:

- The number of required differences to achieve stationarity was determined using `pmdarima`'s `ndiffs()` function.
- Stationarity was verified with the Augmented Dickey-Fuller (ADF) test.
- The number of lags for the Auto-Regressive (AR) component was selected using the Partial Autocorrelation Function (PACF) plot.
- The number of lags for the Moving Average (MA) component was selected using the Autocorrelation Function (ACF) plot by identifying the lag where autocorrelation dropped below the significance threshold.
- A SARIMAX model with a seasonal period of 52 (corresponding to 52 weeks in a year) was used to fit the data.
- In the same way the seasonal variables P,D,Q were obtained using a differencing of 52 and repeating the above steps.

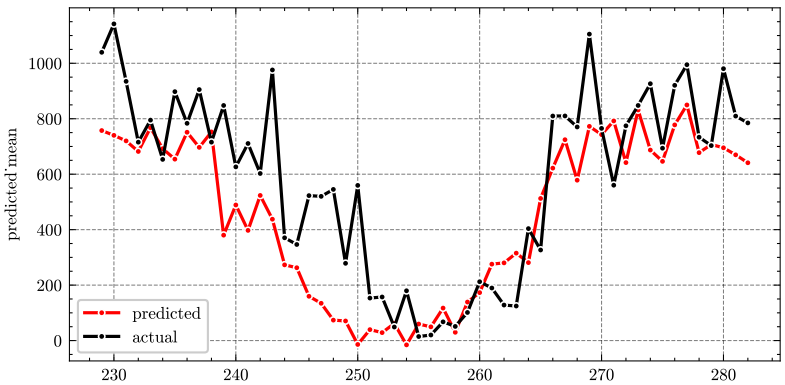

Fig. 2 Actual vs Predicted for the SARIMAX Model

6.2.2 LSTM (Long Short-Term Memory):

The LSTM model was designed to capture long-term temporal dependencies in the time-series data. The data was structured such that each input sequence consisted of 52 consecutive energy readings, and the target output was the 53rd reading.

The model architecture is as follows:

- **Input Layer:** Sequence input of shape (52, 1) representing weekly energy readings.
- **First LSTM Layer:** 700 units, `tanh` activation, with `return_sequences=True`.
- **Dropout Layer:** 20% dropout for regularization.
- **Second LSTM Layer:** 500 units with `tanh` activation.
- **Dropout Layer:** Another 20% dropout.
- **Fully Connected Dense Layers:**
  - Dense Layer 1: 100 neurons
  - Dense Layer 2: 50 neurons
  - Dense Layer 3: 25 neurons
  - Dense Layer 4: 12 neurons
- **Output Layer:** Single neuron for predicting the next time-step value.

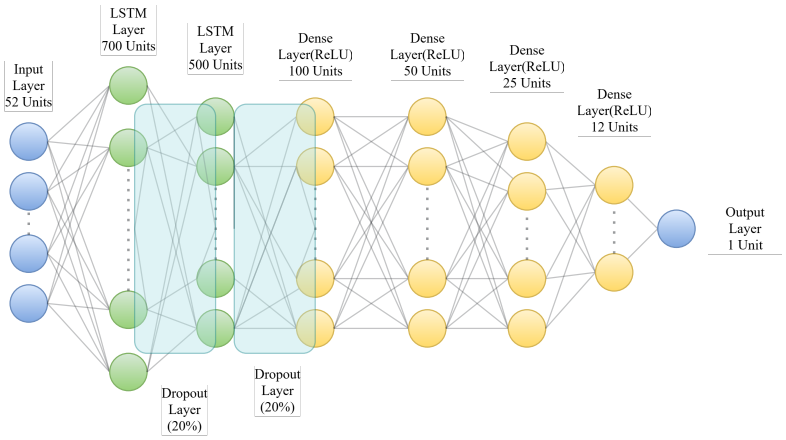

Fig. 3 LSTM Architecture Used

This model was trained until `EarlyStopping` came into play and stopped the model from overfitting when it started observing `validation loss` increase.

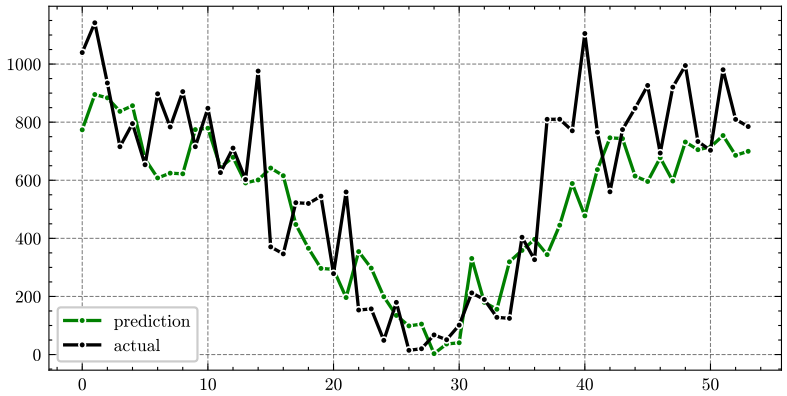

Fig. 4 The Predicted vs Actual value for the LSTM Model

6.2.3 Transformer

The Transformer model was implemented using PyTorch to predict weekly PV generation based on a window of 25 past observations. Unlike RNNs or LSTMs, the Transformer processes all time steps in parallel using self-attention mechanisms, which helps it capture long-range dependencies in time-series data efficiently.

Data Preparation: The dataset was first scaled using a MinMaxScaler and split into training and testing sets. Sequences of 25 time steps were extracted to form the input samples, and the 26th value was used as the prediction target. A custom `PVGenDataset` class was used to structure the data appropriately for the model.

The Transformer consists of the following components:

- **Input Linear Layer:** Projects 1D input into a higher-dimensional embedding space of size $d_{\text{model}} = 64$.
- **Learnable Positional Encoding:** A parameter of shape (25, 64) is added to the input embeddings to retain temporal order information.
- **Transformer Encoder:** Composed of two encoder layers with four attention heads each and dropout of 0.1, implemented using `nn.TransformerEncoder`.
- **Output Layer:** A final linear decoder maps the last time step's embedding to a scalar output representing the predicted PV value.

Fig. 5 Transformer Architecture Used

The model was trained using the Adam optimizer with a learning rate of 0.001 and Mean Squared Error (MSE) loss for 30 epochs. A batch size of 64 was used during training. The training loop was wrapped with a progress bar using `tqdm` for real-time tracking. This architecture benefits from the parallelism and global attention mechanisms inherent to Transformers, making it effective in modeling non-linear and long-range patterns in solar energy generation data.

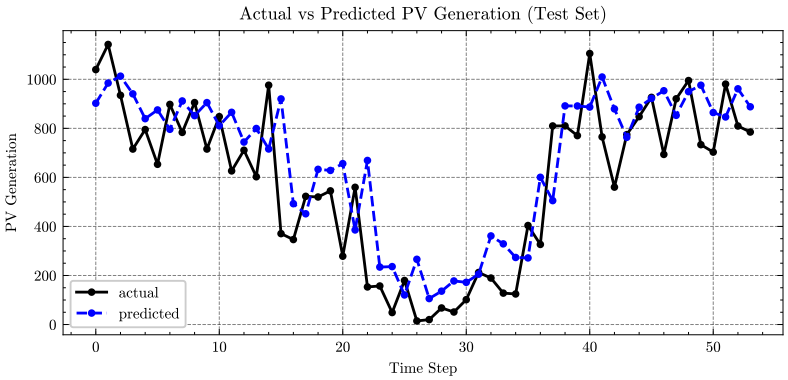

Fig. 6 The Actual vs Predicted values for the Transformer Model

6.2.4 VAR (Vector Autoregression)

The Vector Autoregression (VAR) model is a statistical approach used for multivariate time-series forecasting. It captures linear interdependencies among multiple time-series variables by modeling each variable as a function of its past values, as well as the past values of other correlated variables.

In our implementation, each variable was modeled using the past 25 values of both itself and the other variable. We used *dayLength* (length of day) as the second variable,

since the energy delta showed strong correlation with it. The VAR model was then trained on the prepared data to capture the dependencies among multiple time series variables for forecasting.

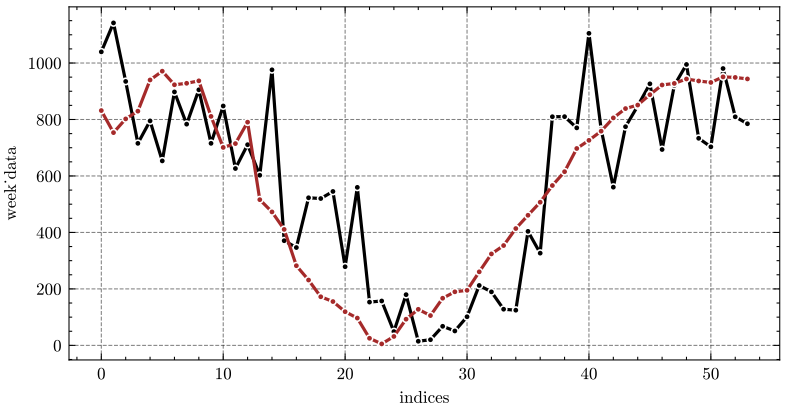

Fig. 7 The Actual vs Predicted values for the VAR Model

7. Ensembling

Ensembling is a powerful technique in machine learning where multiple models are combined to improve performance. The technique used here is **Bagging** (Bootstrap Aggregating), which involves training several models in parallel and then averaging their predictions to reduce variance.

The notebook implements three key ensemble strategies:

- **Multi-Layer Perceptron (MLP)**
- **Convolutional Neural Network (CNN)**
- **Simple Averaging Ensemble**

7.1. Data Preparation

The predictions from four base models — LSTM, Transformer, VAR, and SARIMA — were consolidated into a matrix of shape $(54, 4)$. This matrix was then split into training and testing datasets to train the three ensemble models.

7.1.1 Multi-Layer Perceptron (MLP) Ensemble

Overview MLP is a type of artificial neural network that consists of multiple layers of neurons, including an input layer, hidden layers, and an output layer. It is commonly used for regression and classification tasks.

Implementation in the Notebook

- The dataset is pre-processed and split into training and testing sets.
- An MLP model is defined using `TensorFlow/Keras`.
- The model consists of multiple dense layers with ReLU activation functions.
- It is trained using Mean Squared Error (MSE) as the loss function.
- Predictions from the MLP are used as part of the ensemble to improve accuracy.

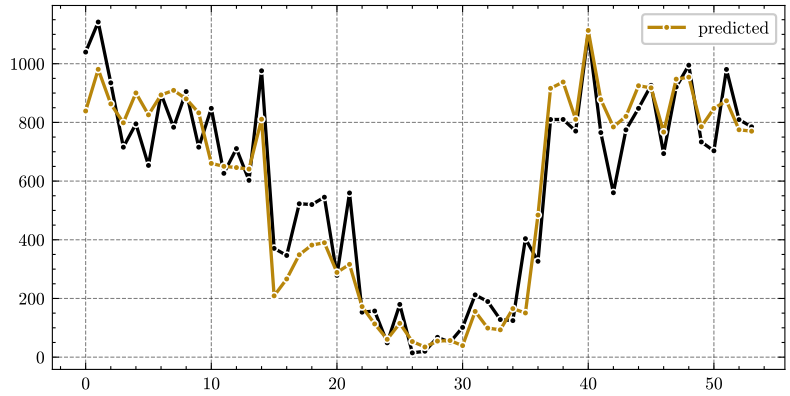

Fig. 8 Result of the MLP Ensemble

7.2. Convolutional Neural Network (CNN) Ensemble

Overview: CNNs are powerful deep learning models that excel at capturing spatial patterns. While traditionally used for image processing, they can also be effectively applied to time series and structured data.

Fig. 9 CNN Model Architecture

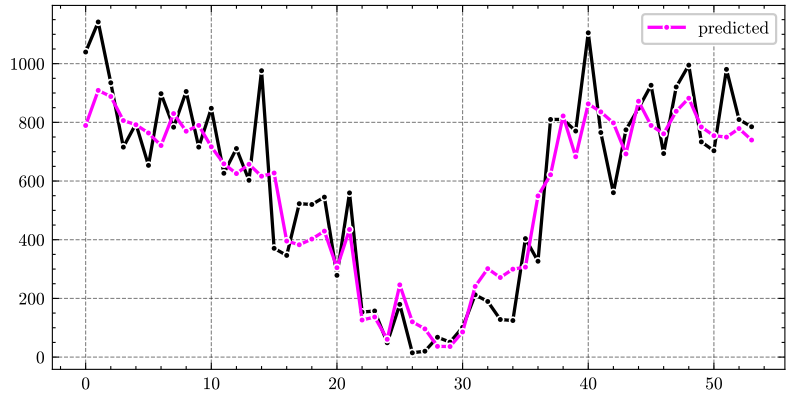

Fig. 10 Results of the CNN Ensemble

Implementation in the Notebook

- The CNN architecture consists of convolutional layers, pooling layers, and fully connected layers.
- Filters and kernels are used to extract features from the input data.
- The model is trained on the dataset using backpropagation and the Adam optimizer.

7.3. Simple Averaging Ensemble

Overview: Simple averaging is one of the most intuitive ensemble methods. It combines multiple models by taking the average of their predictions, which helps in reducing variance and improving stability.

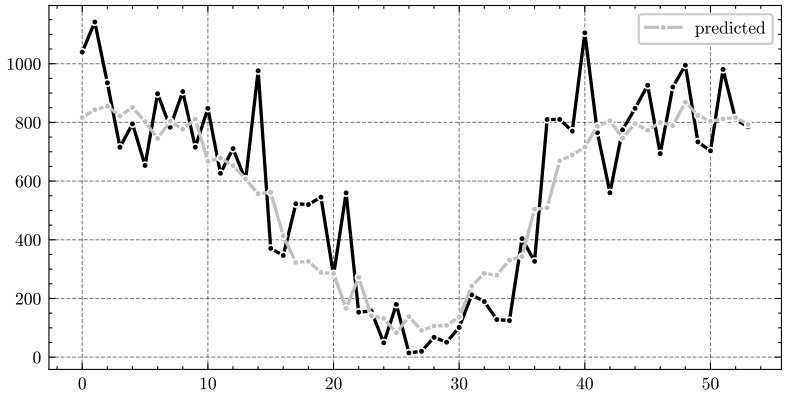

Fig. 11 Simple Mean Ensembling Result

Implementation in the Notebook

- Predictions from different models (MLP, CNN, and possibly others) are collected.
- The final prediction is obtained by averaging the output of all the models.
- This ensemble approach smooths out errors from individual models, improving generalization.

8. Conclusion

The notebook effectively employs MLP, CNN, and Simple Averaging ensembles to improve model performance. MLP and CNN capture different aspects of the data, while simple averaging ensures robustness by reducing individual model errors. Together, these techniques create a more reliable and accurate predictive model.

Ensembling remains a crucial strategy in machine learning, particularly for complex datasets where a single model may not perform optimally.

8.1. Final Output and Evaluation

The performance of the ensemble model was evaluated using standard error metrics such as Mean Squared Error (MSE) and Mean Average Error (MAE). The results were visualized using plots to compare the actual vs. predicted values.

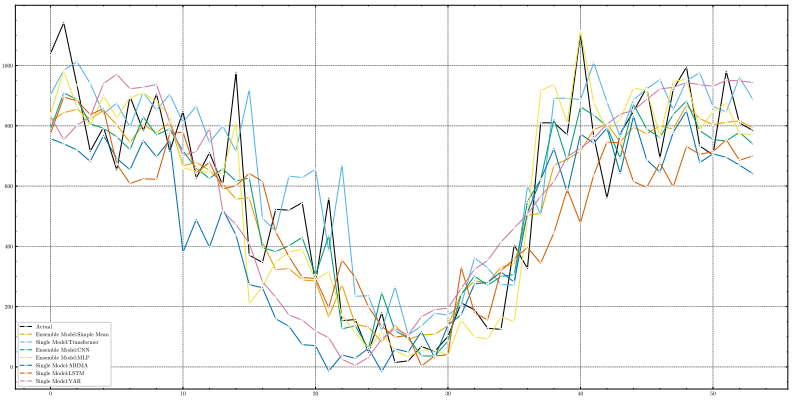

Fig. 12 Final predictions of all models vs actual

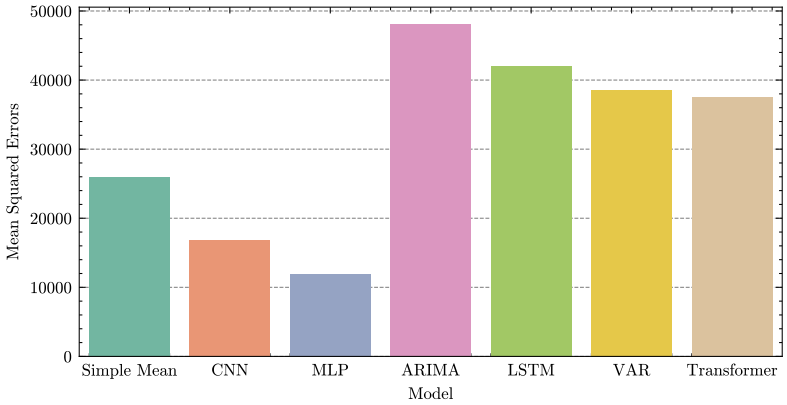

Fig. 13 Comaprison of Mean Squared Errors of different models

- The ensemble model outperformed individual models by reducing prediction errors.
- Visual analysis showed that the ensemble captured trends more effectively.

9. Conclusion

The proposed methodology successfully used deep learning models and ensemble techniques for time-series forecasting. The combination of LSTM,ARIMA,VAR,MLP,Transformer and CNN models improved the prediction accuracy, demonstrating the effectiveness of hybrid forecasting techniques.

10. References

1. Hanuel Eom, Yongju Son and Sungyun Choi, School of Electrical Engineering, Korea University, "Feature-Selective Ensemble Learning-Based Long-Term Regional PV Generation Forecasting".
2. Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory". Neural Comput 1997; 9 (8): 1735–1780.
3. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning internal representations by error propagation. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Vol. 1: Foundations)", MIT Press, 1986.
4. Box, G.E.P., and Jenkins, G.M. "Time Series Analysis: Forecasting and Control." San Francisco: Holden-Day, 1970.
5. Bandara, K., Bergmeir, C., Smyl, S. (2020). Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Systems with Applications, 140*, 112896. https://doi.org/10.1016/j.eswa.2019.112896
6. Borovykh, A., Bohte, S., Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691.*
7. Livieris, I. E., Pintelas, E., Pintelas, P. (2020). A CNN–LSTM model for gold price time-series forecasting. *Neural Computing and Applications, 32*, 17351–17360. https://doi.org/10.1007/s00521-020-04867-x
8. Zhang, G., Eddy Patuwo, B., Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting, 14* (1), 35–62. https://doi.org/10.1016/S0169-2070(97)00044-7
9. Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting, 36* (1), 75–8