Final Report on Time Series Forecasting Project

Objective

The goal of this project was to predict future temperature values using IoT sensor data. We experimented with various machine learning and deep learning models, including traditional statistical methods like ARIMA, machine learning models such as Random Forest, and deep learning approaches like Artificial Neural Networks (ANN) and LSTM. We also implemented hyperparameter tuning and added an attention mechanism to further improve the results.

1. Rationale Behind Model Choices and Design

Given that temperature data exhibits time dependencies, we approached this task as a time series forecasting problem. To capture both short-term fluctuations and long-term trends, we used a diverse set of models:

- **ARIMA:** A well-known statistical model designed for univariate time series data. It's a traditional approach that handles seasonality and trends reasonably well for stationary time series.

- **Random Forest Regressor:** A machine learning model capable of capturing complex relationships between features but often overlooks temporal dependencies.

- **Artificial Neural Networks (ANN):** A feedforward neural network was used to capture potential non-linear relationships between features and target values.

- **LSTM (Long Short-Term Memory):** LSTM networks are specifically designed to model sequential data and handle long-term dependencies, making them suitable for time series forecasting.

- **XGBoost (stacking):** XGBoost is an ensemble method that was explored as a possible combination of both feature-based and sequential dependencies combining the outputs of LSTM and ANN.

- **Categorised into "in" and "out":** The rationale for using separate models for "In" and "Out" temperature categories is driven by the assumption that the patterns influencing temperature changes in indoor environments differ significantly from those in outdoor environments. By separating them, each model can be optimized to capture the specific characteristics of these environments. This design choice resulted in strong R2 scores for both the "In" and "Out" models

2. Experiments Conducted and Their Impact

Sub-Experiment 1: Hyperparameter Tuning

We tuned parameters like learning rate, batch size, and the optimizer for both ANN and LSTM models. The findings were as follows:

- **Learning Rate:** A smaller learning rate (0.001) provided better results than a larger one, avoiding overshooting the minimum during optimization.

- **Batch Size:** A batch size of 64 proved optimal for balancing computational efficiency and model performance.

- **Optimizer:** Adam optimizer worked best for both models compared to SGD and RMSProp.

Results: Hyperparameter tuning improved the model's performance by reducing loss during training and validation.

Sub-Experiment 2: Model Architecture Enhancement

For the ANN model, we experimented by adding additional layers and neurons. Increasing model depth and adding more neurons per layer generally improved performance, but also increased the risk of overfitting, which was mitigated using dropout layers.

Results: Adding more layers improved model accuracy but required more regularization techniques like dropout to avoid overfitting.

Sub-Experiment 3: Attention Mechanism (LSTM)

We added an attention layer to the LSTM model to allow the model to focus on specific time steps when making predictions. This enhanced the ability to capture important patterns in the data.

Results: The attention mechanism provided a small but noticeable improvement in prediction accuracy, particularly in handling complex temporal dependencies.

Sub-Experiment 4: Traditional Model ARIMA

ARIMA was used as a baseline model. While ARIMA captures short-term dependencies well, it struggled with more complex patterns.

Results: The ARIMA model performed poorly compared to machine learning and deep learning models, with higher test RMSE.

3. Lessons Learned

- **Model Choice:** Deep learning models like LSTM with attention mechanisms outperformed traditional models such as ARIMA and machine learning models like Random Forest and XGBoost. Categorised model building depicted better results as well.

- **Feature Engineering:** Including relevant temporal features like year, month, hour, etc., had a noticeable impact on the model’s performance, especially for non-sequential models like Random Forest.

- **Hyperparameter Tuning:** Tuning the hyperparameters, particularly for deep learning models, was essential to improve model performance and avoid overfitting.

- **Regularization:** Techniques like dropout and early stopping were critical in preventing overfitting in neural networks.

4. Potential Next Steps

- **Incorporate more advanced feature engineering** to capture seasonality and other latent variables.

- **Explore hybrid models** that combine LSTM's ability to model sequential data with models like XGBoost for better generalization.

- **Retrain the model periodically** to handle new incoming data, especially in real-time IoT environments.

- **Implement model versioning and checkpoints** to track and resume training efficiently without redoing from scratch.

- **Deploy the model using time-series forecasting libraries** like `Prophet` in production to benefit from scalability and operational efficiency.