ITAI 3377

Professor Patricia McManus

Hoang Dinh

**L06 Final Report**

1. **Data Preparation and Preprocessing**

The dataset used in this project consists of temperature readings from multiple rooms across different time intervals. The initial preprocessing involved:

* Loading and cleaning the data using pandas.
* Standardizing column names to `unique\_id`, `ds` (timestamp), and `y` (target variable).
* Parsing timestamps with day-first formatting.
* Sorting data by room and time to maintain proper time-series order.
* Checking for missing values — no missing entries were found.
* Outlier detection using Z-score filtering (threshold of 3) — no significant outliers were detected.
* Normalization of the target variable using MinMaxScaler to scale temperatures between 0 and 1. This step improved convergence for the generative model and ensured stable forecasting.

1. **Model Selection and Training (Nixtla’s MLForecast)**

For the forecasting task, we leveraged Nixtla’s `mlforecast` library, which provides automated time-series modeling on tabular machine learning frameworks.

* Model Used: RandomForestRegressor
* Lags Selected: [1, 2, 3, 6, 12] to capture short and mid-term dependencies
* Lag Transforms: A rolling mean was applied to lag 1 (rolling\_mean\_3)
* Date Features: hour, dayofweek were included to model cyclic time behavior
* Frequency: Hourly ('h')

mlforecast automatically engineered features and trained the model across grouped time series (one per room). The model was trained on the cleaned dataset and evaluated on a 12-step holdout horizon.

1. **Feature Engineering and Significance of Custom Features**

Nixtla’s framework handled the majority of feature engineering through:

* Lagged variables
* Date-derived features
* Rolling average lag transforms

We initially attempted to incorporate two custom domain-informed features:

1. rolling\_mean\_3: A 3-point rolling average of temperature
2. temp\_change: First difference to capture rate of change

However, these were removed during modeling due to conflicts with mlforecast's internal structure. They were still useful for exploratory analysis and could be reintegrated as dynamic features if needed.

1. **Model Evaluation and Cross-Validation**

Holdout Evaluation:

A 12-step prediction was compared against the latest available data:

|  |  |
| --- | --- |
| Metric | Value |
| MAE | 0.1411 |
| MSE | 0.0220 |
| MASE | 3.4513 |

These metrics provided a strong baseline for comparison.

**\*\*Rolling-Origin Cross-Validation\*\***

A manual CV loop was implemented to simulate multiple train-test splits. However, due to limited testable window size, cross-validation was skipped with a fallback warning:

"⚠️ Not enough data to perform rolling-origin cross-validation."

Reducing the horizon or window size would enable this evaluation in future runs.

1. **Application of Generative Models and Their Impact**

To improve model generalization and simulate data-rich conditions, a Variational Autoencoder (VAE) was implemented using TensorFlow:

* The VAE was trained on sequences of 24-hour temperature windows across all rooms.
* Once trained, the decoder generated 500 synthetic sequences, each 24 time steps long.
* These synthetic records were reshaped and integrated into the training set.

Retraining Results After Augmentation:

|  |  |  |
| --- | --- | --- |
| Metric | Before Augmentation | After Augmentation |
| MAE | 0.1411 | 0.1230 |
| MSE | 0.0220 | 0.0196 |
| MASE | 3.4513 | 3.0079 |

The model improved across all metrics, confirming that VAE-generated data enhanced the learning process and reduced forecasting error.

1. **Individual Reflection**

This project taught me a lot about advanced topics in time-series forecasting and machine learning. I already knew the basics of modeling and working with data, but this project pushed me to:

* Understand how Nixtla’s mlforecast tool works behind the scenes.
* Debug and resolve complex errors, particularly around feature handling and VAE model architecture
* Learn how to implement and integrate generative modeling (VAE) into a forecasting workflow
* Manage model compatibility between frameworks like TensorFlow and scikit-learn
* Analyze and interpret performance improvements driven by synthetic data

One of the toughest parts was fixing errors with custom features in Nixtla and making sure the VAE outputs had the right shape and worked well for improving the model.

By the end of the project, I got hands-on experience with generative models, preparing real-world time-series data, and boosting model performance using data augmentation. These are skills I’m sure will be useful in future AI and industrial IoT projects.

1. **References**

<https://nixtlaverse.nixtla.io/statsforecast/src/core/models.html>

<https://docs.nixtla.io/>

<https://github.com/Nixtla/nixtla>

<https://nixtlaverse.nixtla.io/neuralforecast/docs/tutorials/forecasting_tft.html>