Adeyemi Kolawole

Professor McManus

ITAI 3377

18 MARCH 2025

L06: IIoT Time Series Forecasting Lab

1. Getting Started: Purpose and Overview

This lab explores time-series forecasting on IIoT temperature data, covering the full pipeline: data cleaning, feature engineering, model selection, and training. Although I initially aimed to use Nixtla's TimeGPT-1, import issues led me to switch to Prophet. I also implemented a Variational Autoencoder (VAE) to generate synthetic data and enhance the dataset.

2. Preparing the Dataset: From Raw Data to Ready-to-Use

2.1. Loading the Dataset and Taking a First Look

- **Dataset Source:** I sourced the "IoT Temperature Forecasting" dataset from Kaggle.
- **Reading the File:** Using `pd.read_csv('IOT_temp.csv')`, I brought the dataset into my workspace via Pandas.
- **Initial Inspection:** A quick glance at the data (column names and sample rows) helped me identify the key columns for dates and temperature values.

2.2. Cleaning Up: Handling Gaps and Formatting Time

- **Filling Gaps:** I addressed missing values using forward fill (`df.ffill()`), which maintained the sequence of the data.
- **Parsing Dates:** I converted the date column into a datetime object with `pd.to_datetime(errors='coerce')` to avoid parsing errors. Any rows with invalid dates were dropped.

- **Sorting & Splitting:** I ordered the dataset chronologically and split it into 80% training and 20% testing sets, preserving the temporal structure.

3. Choosing and Training the Right Model

3.1. Weighing the Options: What Model Should I Use?

I initially intended to use Nixtla's TimeGPT-1 for its cutting-edge capabilities, but compatibility issues with the library's import structure pushed me toward Prophet—a well-documented and reliable alternative.

3.2. Building the Forecast with Prophet

- **Installation & Setup:** I installed Prophet (`pip install prophet`) and imported it using `from prophet import Prophet`.

- **Reformatting Data:** The dataset was reshaped to include Prophet's expected columns: `ds` for dates and `y` for temperature.

- **Training the Model:** I instantiated and trained a Prophet model on the training dataset.

- **Making Predictions:** A future dataframe matching the test period was generated, and predictions were made with the `predict()` method.

- **Visualizing Results:** I used Prophet's built-in visualization tools to plot the forecast alongside historical data.

4. Digging Deeper: Creating Useful Features

4.1. Extracting What Matters

- **Built-in Features:** Prophet automatically captures time-series elements like trend and seasonality.

- **Custom Features I Added:**

    - **Lag Features:** I introduced lag variables—`lag_1` (1 period back) and `lag_7` (7 periods back)—to help the model factor in recent history.

    - **Rolling Mean:** I calculated a 3-point rolling average (`rolling_mean_3`) to help smooth the data and emphasize longer-term trends.

4.2. Why These Features Matter

- **Lag Features:** They help the model “remember” recent patterns that might affect future values.

- **Rolling Averages:** These reduce the noise in the data, allowing the model to better detect trends and recurring patterns.

---

5. How Did the Model Perform?

5.1. Measuring Forecast Accuracy

- **Metrics Used:** I evaluated model performance using Mean Absolute Error (MAE) and Mean Squared Error (MSE).

- **How I Did It:** I merged forecasted results with actual test values and used Scikit-Learn’s functions to compute the metrics.

5.2. Thinking Ahead: Why Cross-Validation Matters

Although I didn't implement rolling-origin cross-validation in this lab, I recognize its value and plan to use it in future projects to better assess model performance across varying forecast periods.

6. Expanding the Dataset with Synthetic Data

6.1. Using a Variational Autoencoder (VAE) to Generate New Data

- **Goal:** I built a VAE to generate synthetic temperature data and boost the diversity of my training set.

- **Architecture Overview:**

    - **Encoder:** A stack of dense layers compressed the input into a latent representation.

    - **Decoder:** A mirrored network reconstructed the original data from this compressed form.

- **Training the Model:** I normalized the temperature data and trained the VAE on it.

- **Creating New Data:** I sampled from the latent space and denormalized the outputs to create realistic synthetic data.
- **Why It's Useful:** This augmented data can make the forecasting model more robust by exposing it to a broader range of patterns.

---

7. Personal Takeaways: Reflections on the Process

This lab was both challenging and insightful. Issues like date parsing errors and changing models mid-project pushed me to adapt. By engineering features and experimenting with a VAE, I gained practical experience in both traditional and modern forecasting, highlighting the importance of flexibility and continuous learning in data science.

---

8. Resources and References

- **Pandas:** https://pandas.pydata.org/docs/
- **Prophet:** https://prophet.readthedocs.io/
- **Nixtla (TimeGPT):** https://github.com/Nixtla/nixtla
- **TensorFlow / Keras:** https://www.tensorflow.org/api_docs
- **Scikit-Learn:** https://scikit-learn.org/stable/documentation.html
- **Jupyter & ipywidgets:** https://jupyter.org/, https://ipywidgets.readthedocs.io/en/stable/