Forecasting Project
Technical Documentation
V2.0

Contents

I. Introduction

This project focuses on forecasting electricity consumption levels for a diverse range of clients across various time intervals, utilizing historical data from the "Electricity Load Diagrams 2011-2014 Data Set" accessible through the UCI Machine Learning Repository[1]. To accomplish this, we will harness cutting-edge time series machine learning methodologies, such as the Prophet, Neural Prophet, Temporal Fusion Transformer (TFT), and DeepAR models.

Our team's primary goal is to scrutinize daily average electricity consumption patterns for individual clients, pinpointing discernable trends, fluctuations, and underlying factors that impact consumption rates. By undertaking this analysis, we aspire to empower utility companies to optimize energy production and distribution processes, while concurrently assisting customers in comprehending and regulating their electricity usage.

Accurate predictions derived from this project can equip energy providers with the necessary insights to devise well-informed energy production, distribution, and pricing strategies. Simultaneously, consumers stand to benefit from these forecasts, enabling them to make judicious decisions regarding their energy consumption habits and promoting a more sustainable and cost-effective energy ecosystem.

[1] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014#

II. Data extraction and processing

Our data source for this project is the "Electricity Load Diagrams 2011-2014 Data Set" from the UCI Machine Learning Repository. The original dataset is stored in a ".txt" file with a semicolon ";" as the delimiter, containing 31 variables in total. The first column represents the date and time in the format "yyyy-mm-dd hh:mm:ss," with each record having a 15-minute gap between its preceding and succeeding records. The subsequent columns denote the electricity consumption of individual clients in kW during 15-minute intervals, represented as float values. To avoid ambiguity, we have replaced the comma "," (originally used as a decimal point) with a standard dot ".".

The dataset encompasses electricity consumption records from 2011-01-01 to 2015-01-01, totaling 140,256 observations. All time labels correspond to Portuguese hours. As our goal is to predict consumption in 2015, we discard the record for 2015-01-01 during data processing. This dataset does not contain duplicate records, missing values, or extreme outliers. However, there are some minor issues to address. Certain clients were established after 2011, resulting in zero consumption for that year. To circumvent this problem, we focus on the time range between 2012-01-01 and 2014-12-31.

Additionally, we identified 40 users who opened their accounts very late in the time frame, which could lead to potential inaccuracies in our analysis. Consequently, we exclude these users from our dataset, resulting in a refined dataset consisting of 105,216 rows and 320 columns.

III. Variable Analysis

1. Target Variables

In this project, our primary focus is on a single target variable: the daily average of electricity consumption, measured in kilowatts (kW). The equation for calculating the daily average is as follows:

$$Daily\ Average\ =\ \frac{\sum daily\ usage\ per\ client}{\#\ of\ active\ clients}$$

It is crucial to exclude users who opened their accounts after January 1st, 2022, from this calculation to ensure accuracy. To accomplish this, we assume that each client's account opening date corresponds to the date they began consuming electricity. Any recorded consumption of 0 kW after the account opening date is considered a day when the client did not consume any electricity. By applying this approach, we can accurately measure the daily average electricity consumption while accounting for clients who joined the dataset at different times.

2. Predictive Variables

In this project, the primary predictive variable is the timestamp, which serves as the foundation for our forecasting efforts. To improve the forecasting performance of the Prophet and Neural Prophet models, we will integrate holiday information as an additional input. The inclusion of this supplementary data enables the models to better account for fluctuations in electricity consumption patterns that occur during holiday periods.

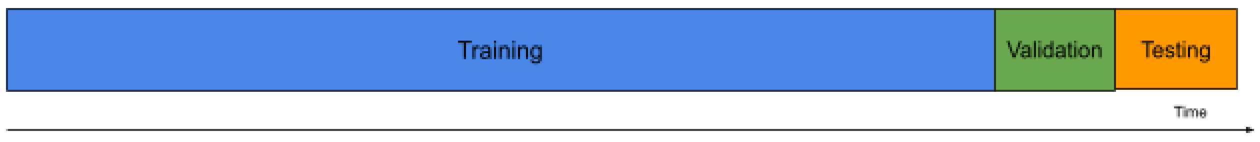
By incorporating holiday information, the models can more effectively capture the nuances of electricity usage trends and the impact of holidays on consumption. This leads to a more comprehensive understanding of the underlying patterns and, ultimately, more accurate and reliable predictions.

Exploiting the ability of the Prophet and Neural Prophet models to accommodate holiday data ensures that our forecasting models are well-equipped to address the complexities of electricity consumption. This added contextual information enhances the models' ability to generalize across various scenarios, resulting in forecasts that are better suited for real-world applications and decision-making processes.

3. Division of Train, Validation, and Test Set

We divide the dataset into three distinct portions - training, validation, and testing - with a ratio of 8:1:1 for each respective segment.

As illustrated in the figure below, 80% of the data is utilized for training the model, while approximately 10% serves as validation data during hyperparameter tuning and model selection. The remaining 10% of the dataset is designated as testing data, employed to calculate the error of the best-performing model. We further subdivide the testing data into three parts, evaluating each segment individually to ensure a thorough assessment of model performance.



IV. Model Training & Prediction Accuracy Analysis

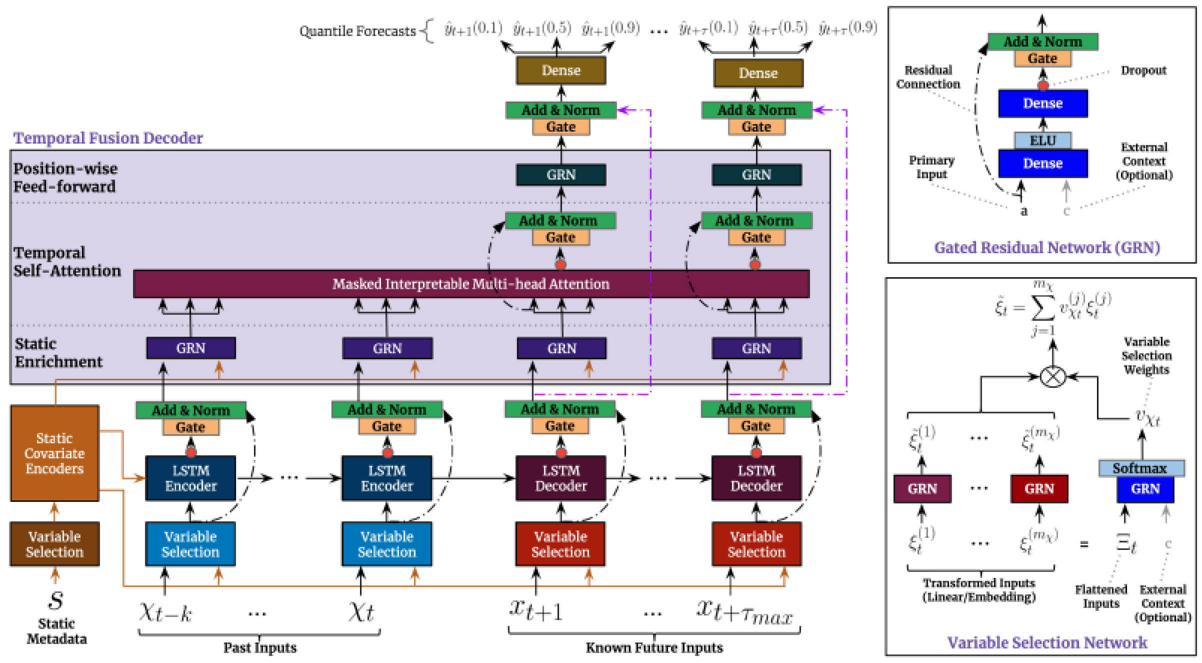
1. Models Introduction

Upon finalizing the candidate predictive variables during the pre-modeling phase, our team diligently executed the requisite data mining and statistical iterations to design and construct the algorithmic solutions. This comprehensive process allowed us to develop robust and accurate forecasting models tailored to the specific challenges presented by our dataset.

- **Facebook Prophet**[2]
  - Prophet is a forecasting method designed for time series analysis, employing an additive model that accommodates non-linear trends along with yearly, weekly, and daily seasonality, as well as holiday effects. This approach is particularly effective for time series exhibiting significant seasonal patterns and with multiple seasons of historical data available. Prophet's robustness extends to handling missing data, trend shifts, and effectively managing outliers, making it a versatile and reliable tool for time series forecasting.
- **Neural Prophet**[3]
  - Neural Prophet is a user-friendly framework designed for interpretable time series forecasting. Built on PyTorch, it seamlessly integrates neural networks with traditional time series algorithms, drawing inspiration from Facebook's Prophet and AR-Net. This powerful combination offers an accessible and efficient approach for predicting temporal patterns while maintaining interpretability and performance in various forecasting applications.

[2] https://facebook.github.io/prophet/
[3] https://neuralprophet.com/contents.html#

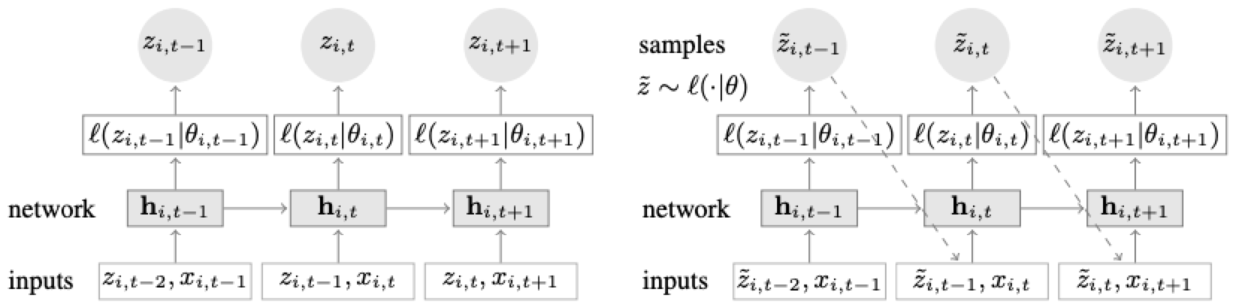
- **Temporal Fusion Transformer**[4]



- ○ Temporal Fusion Transformer (TFT) is a cutting-edge deep learning model designed to tackle time series forecasting challenges. Developed by Google Research, TFT combines the strengths of recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformers to effectively capture the data's complex temporal dependencies, trends, and patterns.
- ○ TFT's flexibility and scalability make it suitable for a wide range of applications, including demand forecasting, energy consumption prediction, and financial market analysis. Its capacity for handling large-scale datasets with diverse features and varying degrees of seasonality contributes to its growing popularity in the field of time series forecasting.

[4] https://arxiv.org/abs/1912.09363

- **DeepAR**[5]



  - DeepAR is a state-of-the-art probabilistic forecasting model developed by Amazon Web Services (AWS) to address time series prediction tasks. This sophisticated deep learning model leverages recurrent neural networks (RNNs) to capture complex patterns, trends, and dependencies within time series data. DeepAR is particularly effective in handling large-scale datasets containing multiple time series with varying degrees of seasonality, noise, and missing values.
  - The key advantage of DeepAR is its ability to generate accurate and flexible probabilistic forecasts, which provide point estimates and confidence intervals for predictions. This feature allows users to understand better the uncertainty associated with forecasts, making it an invaluable tool for decision-making in various domains, such as demand forecasting, energy consumption prediction, and financial market analysis.

2. Hyperparameters Selection of Algorithms

- **Facebook Prophet**
  - changepoint_prior_scale: Adjusts the flexibility of the automatic changepoint detection.
  - changepoint_range: Determines the proportion of the historical data considered for detecting potential change points.

[5] https://arxiv.org/abs/1704.04110

- holidays_prior_scale: Regulates the strength of the holiday components in the model.
- interval_width: Specifies the width of the uncertainty intervals for the forecast.
- seasonality_prior_scale: Controls the flexibility of the seasonal components.
- uncertainty_samples: Sets the number of Monte Carlo samples used to estimate the uncertainty intervals.

- **Neural Prophet**
  - batch_size: Determines the number of training samples used in each update of the model weights.
  - changepoints_range: Controls the proportion of the historical data considered for detecting potential changepoints.
  - daily_seasonality: Enables or disables daily seasonality in the model, allowing the model to capture daily patterns if set to True.
  - d_hidden: Specifies the size of the hidden layers in the model's deep learning components.
  - epochs: Sets the number of times the model iterates through the entire training dataset during optimization.
  - growth: Defines the type of trend component used in the model, either linear or logistic, to better fit the data's characteristics.
  - impute_missing: Enables or disables the imputation of missing values in the time series data, which can help the model handle incomplete data.
  - learning_rate: Adjusts the step size used in the optimization process, influencing the speed and stability of model convergence.
  - loss_func: Specifies the loss function used to measure the discrepancy between the model's predictions and the actual values during optimization.
  - normalize: Enables or disables data normalization, which can improve model performance by scaling the input features to a consistent range.

  - num_hidden_layers: Sets the number of hidden layers in the model's deep learning components, affecting the model's capacity and expressiveness.
  - n_change_points: Determines the number of potential changepoints used in the model, allowing for more flexibility in capturing trend changes.
  - n_forecasts: Specifies the number of future time steps to forecast, defining the forecast horizon.
  - optimizer: Selects the optimization algorithm used during model training, which can impact the convergence and performance of the model.
  - seasonality_mode: Defines whether seasonality components are modeled as additive or multiplicative.
  - weekly_seasonality: Enables or disables weekly seasonality in the model, allowing the model to capture weekly patterns if set to True.
  - yearly_seasonality: Enables or disables yearly seasonality in the model, allowing the model to capture yearly patterns if set to True.
- **Temporal Fusion Transformer**
  - context_length: Specifies the number of past time steps used as input to the model.
  - disable_static_features: Enables or disables the use of static features in the model.
  - disable_known_covariates: Enables or disables the use of known covariates in the model.
  - num_layers: Sets the number of layers in the Temporal Fusion Transformer's architecture, affecting the model's capacity and complexity.
  - hidden_size: Determines the size of the hidden layers in the model, which can impact the model's capacity and expressiveness.
  - dropout_rate: Specifies the dropout rate used for regularization, helping to prevent overfitting by introducing noise during training.
  - embedding_dimension: Controls the size of the embeddings used for categorical features.

    - distr_output: Sets the type of distribution used to model the output, allowing the model to account for different features of the target variable.
    - scaling: Enables or disables data scaling, which can improve model performance by transforming the input features to a consistent range.
    - epochs: Adjusts the number of times the model iterates through the entire training dataset during optimization.
    - batch_size: Specifies the number of training samples used in each update of the model weights.
    - num_batches_per_epoch: Sets the number of batches processed in each epoch.
    - learning_rate: Determines the step size used in the optimization process, affecting the speed and stability of model convergence.
- **DeepAR**
    - context_length: Defines the number of past time steps used as input to the model.
    - disable_static_features: Enables or disables the use of static features in the model.
    - disable_known_covariates: Enables or disables the use of known covariates in the model.
    - num_layers: Sets the number of layers in the DeepAR's architecture, affecting the model's capacity and complexity.
    - hidden_size: Specifies the size of the hidden layers in the model, which can impact the model's capacity and expressiveness.
    - dropout_rate: Determines the dropout rate used for regularization, helping to prevent overfitting by introducing noise during training.
    - embedding_dimension: Controls the size of the embeddings used for categorical features.
    - distr_output: Sets the type of distribution used to model the output, allowing the model to account for different features of the target variable.

- scaling: Enables or disables data scaling, which can improve model performance by transforming the input features to a consistent range.
- epochs: Adjusts the number of times the model iterates through the entire training dataset during optimization.
- batch_size: Specifies the number of training samples used in each update of the model weights.
- num_batches_per_epoch: Sets the number of batches processed in each epoch.
- learning_rate: Determines the step size used in the optimization process, affecting the speed and stability of model convergence.

3. Evaluation Metric

One metric was adopted in model evaluation: Mean Absolute Percentage Error

$$M = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

M = mean absolute percentage error

n = number of times the summation iteration happens

A_t = actual value

F_t = forecast value

Mean Absolute Percentage Error (MAPE) quantifies the percentage error between actual and predicted values, making it a suitable choice for evaluating the performance of a time-series model. By calculating the average magnitude of errors relative to the true values, MAPE provides an easily interpretable metric for assessing the accuracy of a model's forecasts.

Model	**Prophet**	**Neural Prophet**	**TFT**	**DeepAR**
Test1	0.03953	0.039054	0.04652	
Test2	0.08385	0.060411	0.13411	
Test3	0.04634	0.035410	0.11127	
Average MAPE	0.05678	0.045012	0.09775	0.24418

4. Algorithmic Solution Finalization

Initially, we execute a basic or "vanilla" version of each forecasting model to establish a baseline performance. Following this, we employ Bayesian optimization techniques to systematically search for the optimal set of hyperparameters for each respective model. This process involves utilizing a validation dataset to fine-tune the hyperparameters and prevent overfitting.

Once the optimal hyperparameter set is identified for each model, we incorporate these values into the respective models and proceed to train the enhanced versions. Throughout the hyperparameter tuning phase, the validation dataset plays a crucial role in guiding the search for the best-performing set and mitigating the risk of overfitting.

Upon completion of the training phase, we evaluate the performance of each forecasting model using the Mean Absolute Percentage Error (MAPE) metric. This measure provides a quantitative means of comparing the models and facilitates the selection of the most accurate and reliable forecasting model for our specific application.

V. Deliverables

1. Results

The results and details of the training process for each forecasting model in the Electricity Forecasting project can be found in the "Model" directory within the GitHub repository. This section provides valuable information on model training, hyperparameter tuning, and performance evaluation for each model. You can access these resources using the link below:

https://github.com/yichuang25/Electricity_Forecasting/tree/main/Model

Within this directory, you will find Jupyter Notebooks for the four forecasting models implemented in the project: Facebook Prophet, Neural Prophet, Temporal Fusion Transformer (TFT), and DeepAR. These notebooks document the model training process, hyperparameter optimization, and the evaluation of their respective performances in electricity consumption forecasting.

By exploring the notebooks, you can gain insights into the training procedures, model configurations, and the methodologies employed to optimize each model. Additionally, you can review the results, such as performance metrics and visualizations, which allow for a comprehensive comparison of the models and an understanding of their strengths and weaknesses in the context of electricity consumption forecasting.

2. Model

The trained forecasting models for the Electricity Forecasting project are conveniently stored in the "SavedModels" directory within the GitHub repository. This allows for easy access and deployment of the models for various applications. You can access the saved models using the link below:

https://github.com/yichuang25/Electricity_Forecasting/tree/main/Model/SavedModels

Within this directory, you will find four distinct forecasting models that have been implemented and saved for this project: Facebook Prophet, Neural Prophet, Temporal Fusion Transformer (TFT), and DeepAR. Each model has its unique strengths and characteristics, allowing for a comprehensive comparison and evaluation of their respective performances in electricity consumption forecasting.

By examining the saved models, you can gain insights into the model architectures, hyperparameter configurations, and other details that contribute to their predictive capabilities. Additionally, you can utilize these pre-trained models to generate forecasts, analyze their performance in different scenarios, or even fine-tune them further using additional data.

3. Datasets

The raw and processed datasets for the Electricity Forecasting project can be found in the dedicated "Data" directory within the GitHub repository. This directory contains both the original data and the cleaned versions, which have undergone preprocessing to ensure quality and consistency. You can access these datasets using the link below:

https://github.com/yichuang25/Electricity_Forecasting/tree/main/Data

For insight into the data exploration and preprocessing methods employed in this project, you can refer to the "Preprocessing" directory. This section of the repository contains the scripts and notebooks used for data cleaning, transformation, and feature engineering. These resources provide valuable information on how the raw data was prepared for use in the forecasting models. Access the preprocessing materials through the following link:

https://github.com/yichuang25/Electricity_Forecasting/tree/main/Preprocessing

By exploring these directories, you can gain a deeper understanding of the data handling process, the steps taken to ensure data quality, and the techniques used to prepare the data for modeling and forecasting.

4. GitHub Repository[6]

The entire project, including all code, data, and documentation, is stored and maintained in a GitHub repository. This centralized storage allows for seamless collaboration, version control, and easy access to the project files by team members and interested parties. You can access the Electricity Forecasting project repository by visiting the following link:

https://github.com/yichuang25/Electricity_Forecasting

Within the repository, you will find directories containing the various components of the project, such as the data preprocessing scripts, the implementations of the forecasting models, and the evaluation metrics. Additionally, there may be supplementary files, such as a README file that provides an overview of the project's structure and guidelines for setup and usage. By exploring the repository, you can gain valuable insights into the project's development, progress, and outcomes.

[6] To access the deliverables, email us for access.

VI. Team information

Haolong Liu - hl3614@columbia.edu

Yichen Huang - yichen.huang@columbia.edu

Taichen Zhou - tz2555@columbia.edu