Title

Machine Learning Applications in Day-Ahead Electricity Price Forecasting

Author

Bell, Vincent

Publication Date

2024

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Machine Learning Applications in Day-Ahead Electricity Price Forecasting

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Vincent Matthew Bell

2024

ABSTRACT OF THE THESIS

Machine Learning Applications in Day-Ahead Electricity Price Forecasting

by

Vincent Matthew Bell
Master of Science in Statistics
University of California, Los Angeles, 2024
Professor Yingnian Wu, Chair

This study reviews machine learning based time series forecasting approaches and applies them to the day-ahead electricity price forecasting problem. We review the literature on transformer models for time series forecasting, in addition to two MLP-based methods, NBEATSx and NHiTS, and XGBoost. Four models are chosen and implemented: NBEATSx, NHiTS, Temporal Fusion Transformer, XGBoost. They are tested on electricity price data from the PJM market. We product prediction intervals using adaptive conformal inference. We find that NBEATSx and NHiTS perform best on this dataset. We find that by averaging as few as two models, we are able to significantly improve prediction accuracy.

The thesis of Vincent Matthew Bell is approved.

Oscar Hernan Madrid Padilla

Frederic R. Paik Schoenberg

Yingnian Wu, Committee Chair

University of California, Los Angeles

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Electricity Price Forecasting

Electricity price forecasting is an essential task for consumers and energy producers in electricity markets. Most markets are operated as power exchanges, which involve two-sided trading between electricity producers and consumers. Electricity is sold on a day-ahead two-sided auction, at which time commitments are sold for each of 24 hours the following day. The market clearing price is determined by matching supply bids from energy producers with demand bids from energy consumers. This two-sided market structure ensures that supply and demand are matched at the lowest possible price. Power exchanges generally also use intra-day trading to ensure demand is met in real time.

It is essential in this task for both consumers and producers to have accurate forecasts of electricity prices in order to make competitive bids. Maciejowska et. al. describes several features that make the dynamics of electricity prices unique, most notably that power systems require stability and balance between supply and consumption; and that increasing penetration of renewable energy sources like solar and wind, which are stochastic in nature, add randomness to electricity production[MUW22].

Formally, let us denote the price at day d and hour h $p_{d,h}$. We seek to predict the 24 hourly prices for a day d and hours $h = 1, ..., 24$ with information available at day $d - 1$. It is common to use past values of day-ahead prices as input to the model, as well as covariates such as load forecasts or weather forecasts. These forecast covariates are generally known into the future at the time of prediction. It is also common to include variables representing calendar features such as hour day of the week, or month. These are also clearly known into

the future at the time of prediction.

Our goal is to predict future hourly prices for day d and hours $h = 1, ..., 24$ using some number of previous values of p, $p_{t1:d-1}$ and exogenous variables $x_{t1:d}$. In this case we can represent the model generally as

$$p_{d,h} = f(p_{t1}, ...p_{d-1}, x_{t1}, ..., x_d, \theta_h) + \epsilon_{d,h}$$

Lago et. al. [LMD21] propose using price lags form days $d-1, d-2, d-3, d-7$, covariates form day $d, d-1, d_7$. This model is

$$p_{d,h} = f(p_{d-1}, p_{d-2}, p_{d-3}, p_{d-7}, x_d, x_{d-1}, x_{d-7}, \theta_h) + \epsilon_{d,h}$$

where x represents the covariates known into the future, θ is the model parameter and $\epsilon_{d,h}$ is an error term. Lago et. al. [LMD21] identify two state of the art benchmark models: the Lasso Estimated AutoRegressive (LEAR) model, which is an auto-regressive model that performs variable selection using the Lasso approach, and a DNN model consisting of a simple neural network. Lago et. al.[LMD21] additionally identify several techniques to consider for improving a models effectiveness. The first approach is model recalibration. In the real-world setting, one prediction of 24 hour values is made per day, and it is desirable to use the most recently available historical data to train the model. For that reason, Lago et. al. demonstrate that it is advantageous to retrain the model for each prediction day. The second modeling choice to make is to the size of recalibration window. While we may have many years of historical data available, market dynamics may change quickly, and data representing past market dynamics might not be useful to predicting current dynamics. There is a trade off between the improved accuracy gained by using larger training dataset and the potential accuracy lost by using data that is irrelevant to current market dynamics. Lago et. al. test models using a range of calibration windows, demonstrating that window size does impact model performance. The third topic is model ensembling. It is well known that ensembling forecast models often improves forecast performance. The reason for this is that model ensembling is a form of regularization, and it may reduce prediction variance. The

final topic of consideration identified is the increasing interest in moving from point predictions to probabilistic forecasts[MUW22]. There are a number of approaches for probabilistic forecasting used in the EPF literature, including distributional neural networks[MNW23], quantile regression and quantile regression averaging[UW21], CRPS regression[BZ24], and conformal prediction [MUW22] [KZ21].

CHAPTER 2

Background: Machine Learning Models for Forecasting

2.1 Transformers for Time Series Forecasting

The Transformer, introduced by Vaswani et. al. in 2017[VSP17], is a neural network architecture that brought about a breakthrough in natural language processing and sequential modeling. Prior to that point, the state of the art for sequential modeling were recurrent neural networks, which were limited due to their sequential architecture, with limited opportunities for parallelized training and running up against memory bottlenecks [VSP17]. The Transformer model replaces the sequential architecture of RNNs with the self-attention mechanism, which computes pairwise measures of importance between elements of a sequence [VSP17]. This ability to be trained in parallel and the shorter path between long-range dependencies contribute to Transformers' great success in sequential modeling [VSP17]

Since the Transformer's inception, researchers have been interested in applying the Transformer to time series forecasting problems [WZZ22]. In particular, Transformers have demonstrated powerful capabilities for long-term time series forecasting, which involves forecasting many steps into the future at one time. We review below many key contributions to the application of Transformers to time series forecasting. Each model below proposes some modification to the Transformer to improve its effectiveness for time series forecasting.

We identify two themes among these modifications. Firstly, there are a number of approaches introduced to improve the memory complexity of self-attention. Because self-attention involves the computation of values for each pair of elements in a sequence, its

memory and time complexity for a single attention head is $\mathcal{O}(L^2)$ if a sequence has length L. The LogSparse Transformer and the Informer propose methods for introducing sparsity in the attention matrix, thereby constraining the memory complexity of the models. The Autoformer reduces memory complexity by selecting the subsequences with the highest autocorrelation values. The FEDFormer reduces complexity by computing attention in the frequency domain.

The second theme among these approaches is the development of methods to capture local contextual information of a time series. While dot product attention works well in the natural language processing setting, it necessarily loses temporal information due to its permutation invariant[ZCZ23]. Several new approaches have been proposed to address this. The LogSparse Transformer and the Informer apply convolution to the inputs of self-attention. The Autoformer and FEDFormer use novel attention-like mechanisms based on principles of classical time series analysis. The Autoformer uses autocorrelation and trend-seasonality decomposition to model the trend and seasonality components separately, and the FEDFormer uses Fourier transform to capture information in the frequency domain. Finally, the Temporal Fusion Transformer uses an LSTM encoder to enhance local learning while using multiple gating and variable selection components to select meaningful inputs.

2.1.1 Attention

Attention is a mechanism that allows a model to learn to select parts of the input sequence, and how important each part is for the modeling task. We can think of attention as analogous to a database lookup, where, given a key and value pair (k, v), and a query q, we have

$$\text{Attention}(q, k, v) = \sum_{1}^{m} \alpha(q, k_i) v_i$$

where $\alpha(q, k_i)$ is an attention weight function that measures the relevance of keys k_i to query q. Then $\sum \alpha(q, k_i) v_i$ produces a weighted version of v according to how relevant k_i is to q for each v_i. In general, it is desired for the attention weights $\alpha_i = \alpha(q, k_i)$ to obey $0 \leq \alpha_i \leq 1$

and $\sum_i \alpha_i = 1$, so that $\sum_{i=1}^m \alpha(q, k_i)v_i$ is a convex combination of values. For any function $a(q, k_i)$ that measures similarity between q and k_i, we can ensure the desired property is met by applying the softmax transformation to a. The exponent function ensures the values are nonnegative, and normalizing each term ensures they sum to 1.

$$\alpha(q, k_i) = \frac{\exp a(q, k_i)}{\sum_i \exp a(q, k_i)}$$

In theory, then, we are free to choose any function a that gives us a meaningful measure of relevance or similarity of q for k. For instance, one natural choice might be to use a kernel function. In that case, the attention function becomes equivalent to a kernel regression estimate where the key-value pairs $\{(k_i, v_i)\}_{i=1}^m$ function as training data, and our prediction is evaluated at input q and is given by

$$\hat{y} = \sum_{i=1}^{m} \frac{\alpha_i}{\sum_{j=1}^m \alpha_j} v_i$$

Kernel functions do not scale well in higher dimensions or with large datasets, and they are not typically used in transformer models. However, we will see below with the Informer that this intuition behind attention is used to motivate the construction of a novel sparse form of attention.[ZLL21]

Instead of kernel-based attention weight functions, the standard attention function used in Transformers is scaled dot-product attention, due to its relative computational efficiency and evident performance. Zhang, et. al. [ZLL21] demonstrate that dot product attention is a very similar, and slightly better, alternative to Gaussian kernel as attention weight function, when layer normalization is used. The Gaussian kernel function can be expanded

$$a(q, k_i) = -\frac{1}{2}\|q - k_i\|^2 = q^T k_i - \frac{1}{2}\|k_i\|^2 - \frac{1}{2}\|q\|^2$$

The last term does not help us measure the relationship between q and k_i because it only depends on q, so it can be dropped; and when layer normalization is used, $\|k_i\|^2$ is a constant, so it can also be dropped. We are then left with the dot product. In scaled dot-product attention, the dot product is scaled by $\frac{1}{\sqrt{d_k}}$ to stabilize the variance and to help ensure

smoother gradients. Thus we have the scaled dot product attention function as it is used in Transformer models.[ZLL21]

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

2.1.2 Transformer

In this section we introduce the Transformer model of [VSP17], which we refer to below as the Vanilla Transformer. The Transformer uses multi-headed self attention along with feed-forward networks in an encoder-decoder structure to predict the next token of sequential data. Originally designed for natural language processing tasks, the Transformer can readily be applied to time series forecasting tasks.

The input $X = (x_1, ..., x_t)^T$, where x_i has dimension d_X is enriched with positional encoding

$$PE_{t,d} = \begin{cases} \sin\left(\frac{t}{10000^{2i/d}}\right) & d = 2i \\ \cos\left(\frac{t}{10000^{2i/d}}\right) & d = 2i+1 \end{cases}$$

$d = 1, ..., d_X$. The positional encoding of X is thus the same dimension as X is added to X to form the input to the transformer.

$$X_0 = X + PE(X)$$

The Transformer architecture consists of an encoder and decoder component. The network takes as input a sequence, such as of word embeddings of a string of text, and outputs softmax scores representing the relative weight assigned to the predicted next word in the sequence. In its original formulation, Transformer predicts tokens recursively, in that its prediction at time t is added to the sequence used as input for the next prediction step.[VSP17]

The encoder component consists of a fixed number of identical layers (in the case of the original Transformer, this number is 6), each consisting of a Multi-Headed Self Attention component and a feed-forward network. Each sublayer is connected with residual connections

and layer normalization. The first component of an encoder layer is Multi-Headed Self Attention, where for each head, the queries, keys and values are projected into a subspace by weight matrices learned as parameters by the network. Attention is computed at each of those projections, and their output is combined by a final parameterized matrix W^O. Multi-headed attention allows the model to learn different dynamics at each head, and combines their output. Layer normalization and residual connection are then applied to the output of the multi-headed self attention component. The second component of an encoder layer is a fully connected feed forward network, and layer normalization and residual connections are applied to its outputs as well. [VSP17]

Thus the encoder component of the Transformer is given by

$$X_{en}^{l,1} = \text{LayerNorm}(X_{en}^{l-1} + MHSA(X_{en}^{l-1}))$$

$$X_{en}^{l,2} = \text{LayerNorm}(X_{en}^{l,1} + FFN(X_{en}^{l,1}))$$

for $l = 1, ..., N$ (eg, $N = 6$), with

$$\text{Multi-Headed Attention} : MHA(Q, K, V) = \text{Concat}(h_1, ..., h_h)W^O$$

Where the ith attention head, h_i

$$h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

W_i^Q, W_i^K, W_i^V, W^O are weight matrices that are learned as parameters by the model.

$$\text{FFN}(X) = \text{ReLU}(XW_1 + b_1)W_2 + b_2$$

The decoder component similarly consists of a fixed number of layers (eg $N = 6$), each consisting of sub-components. In addition to the multi-headed self attention and feed-forward network of the encoder component, the decoder first passes its input through mast multi-head self attention, with layer norm and residual connections. Masked attention is used to prevent data leakage, ensuring the model only computes attention weights between a token

and tokens that are prior to it in the sequence. The output of the masked multi-headed self attention is then combined with the output of the decoder layer in a second multi-headed attention component, where the output The decoder component takes as input the target sequence Y, shifted right by one step. The feed-forward network of the second sub-layer is given by

$$Y_0 = Y + PE(Y)$$

$$X_{de}^{l,1} = \text{LayerNorm}(X_{de}^{l-1} + \text{Masked MHSA}(X_{de}^{l-1}))$$

$$X_{de}^{l,2} = \text{LayerNorm}(X_{de}^{l,1} + MHA(Q = X_{de}^{l,1}, K = X_{en}^{N}, V = X_{en}^{N}))$$

$$X_{de}^{l,3} = \text{LayerNorm}(X_{de}^{l,1} + FFN(X_{de}^{l,1}))$$

The model outputs

$$X_{out} = \text{softmax}(\text{Linear}(X_{de}^{N}))$$

The Transformer's superior ability to learn long-range dependencies and to be trained with parallelization has contributed to its success. However, the model's computational complexity due to the need to compute attention scores for every pair of sequence elements, and its difficulty n learning temporal contextual information and limited its success in applications to time series forecasting problems. In the following sections we review modifications proposed to enhance the Transformer's predictive power for time series forecasting.[VSP17]

2.1.3 Log Sparse Transformer

The LogSparse Transformer [LJX19] proposes two separate modifications to the Vanilla Transformer that address specific shortcomings. Firstly it is observed that the point-wise attention in the Vanilla Transformer does not capture local contextual information, which, it is hypothesized, is essential to time series forecasting. To address this, the authors modify point-wise self attention by applying causal convolution to the input queries and keys. [LJX19]

The authors propose modifying the standard multi-headed attention by first applying 1d convolution with kernel size k to the queries and keys,

$$MHA_{conv} = MHA(\text{Conv}_{1,d}(Q), \text{Conv}_{1,d}(K), V)$$

The second issue addressed by the LogSparse Transformer is the memory bottleneck inherent in computing attention, which has memory complexity $\mathcal{O}(L^2)$ for each layer. To address this, they propose a form of sparse attention in which only a select number of pairwise attention scores are computed. If I_l^n is the set of indices of cells for which cell l attends to, the authors seek to reduce $|I_l^n|$ such that $|I_l^n| \propto \log L$. To do this they propose increasing step size exponentially as we move away from cell l:

$$I_l^k = \left\{l - 2^{\lfloor \log_2 l \rfloor}, l - 2^{\lfloor \log_2 l \rfloor - 1}, l - 2^{\lfloor \log_2 l \rfloor - 2}, l - 2^0, l\right\}$$

Thus each layer only performs $L \log L$ attention computations, reducing memory complexity to $O(L \log L)$. Further, the authors prove a result guaranteeing that there is a path in the network between every pair of cells if there are $\lfloor \log_2 l \rfloor + 1$ attention layers, suggesting that introducing sparsity in this way should not prevent the model from being able to learn long-term dependencies.

The authors propose two additional modifications to further optimize the model: restart attention and local attention. With restart attention, after a certain distance from l, the step size goes back to zero and begins increasing by powers of two again. With local attention, we compute attention for each element in some window of steps prior to l, and then begin increasing step size to introduce sparsity.[LJX19]

2.1.4 Informer

The Informer[ZZP21] takes up the same issues as the LogSparse Transformer, and proposes more sophisticated approaches to address them. It aims to select the most important attention pairs using an information-theoretic measure of importance, instead of using a logarithmically decaying sampling frequency. The authors propose a measure of the importance

of an attention weight based on Kullback-Liebler divergence and a method for choosing the approximately most important set of pairs.[ZZP21]

The authors make use of the probabilistic interpretation of attention to propose a measure of importance of an attention weight. We can use the kernel regression interpretation of attention to understand its probabilistic interpretation as an estimate of the conditional mean of v_j

$$\text{Attention}(q_i, K, V) = \sum_j \frac{k(q_i, k_j)}{\sum_l k(q_i, k_l)} v_j = E_{p(k_j|q_i)}[v_j]$$

In the case of scaled dot product attention $k(q_i, k_j) = \exp \frac{(q_i k_j^T)}{\sqrt{d_k}}$. We are interested the underlying distribution implied in this definition of attention, $p(k_j|q_i)$. In the null case where all attention weights are equal, we would have $p(k_j|q_i) = q(k_j|q_i) = \frac{1}{T}$ where T is the length of K.The Kullback-Liebler divergence measures the difference between two probability distributions, and we can use it to measure the difference between the conditional distribution $p(k_j|q_i)$ underlying the scaled dot product attention weight, and the null distribution $q(k_j|q_i) = \frac{1}{T}$. This value is

$$KL(q\|p) = \ln\left(\sum_{l=1}^{T} \exp\left(\frac{q_i k_l^T}{\sqrt{d_k}}\right)\right) - \frac{1}{T}\sum_{j=1}^{T}\left(\frac{q_i k_j^T}{\sqrt{d_k}}\right) - \ln T$$

The authors propose the measure of importance of query q_i, which they call its sparsity measurement, to be this KL divergence measure with the constant term dropped.

$$M(q_i, K) = \ln\left(\sum_{l=1}^{T} \exp\left(\frac{q_i k_l^T}{\sqrt{d_k}}\right)\right) - \frac{1}{T}\sum_{j=1}^{T}\left(\frac{q_i k_j^T}{\sqrt{d_k}}\right)$$

Our goal is to construct $\bar{Q}$, a sparse version of Q, composed by selecting the attention pairs that are most important as measured by $M(q_i, K)$. However, to compute $M(q_i, K)$ for all pairs would not result in a reduction of computational complexity, as we'd have to still compute the dot product for each pair. Additionally, the first term of $M(q_i, K)$ as defined above is numerically unstable. Therefore, we use an approximation method that is

theoretically justified. First the authors prove that

$$\ln L_k \leq M(q_i, K) \leq \max_j\{\frac{q_i k_j^T}{\sqrt{d}}\} - \frac{1}{L_k}\sum_{j=1}^{L_k}\{\frac{q_i k_j^T}{\sqrt{d}}\} + \ln L_k$$

They argue that we can approximately find the top u pairs by selecting the pairs using the following measure

$$\bar{M}(q_i, K) = \max_j\{\frac{q_i k_j^T}{\sqrt{d}}\} - \frac{1}{L_k}\sum_{j=1}^{L_k}\{\frac{q_i k_j^T}{\sqrt{d}}\}$$

where we compute it only using a sample of pairs of size $U = L_k \ln L_Q$. The intuition is that for some fixed u, to choose the top $u = c * \ln L_Q$ (where c is a chosen constant) queries measured by this score, compute their attention weights, and leave the remaining queries' scores to be 0. These represent the queries that are most different from the null case and are thus the most important queries.[ZZP21]

The second major modification to the Vanilla Transformer architecture introduced with the Informer is the use of 1d convolution for learning local contextual information and max pooling for reducing sequence length by half at each layer. This step replaces the feed-forward component of the standard transformer encoder.

$$X_{en}^{l,1} = \text{ProbSparse MHSA}(X^{l-1})$$

$$X_{en}^{l,2} = \text{MaxPool}(ELU(\text{Conv1d}_k(X_{en}^{l,1}))$$

The Informer uses a standard decoder similar to the Vanilla Transformer. However, the Informer predicts the entire horizon sequence at one time, instead of recursively predicting one step at a time. To accomplish this, the start token, consisting of some portion of the input sequence, is padded with a placeholder with length of the desired output. [ZZP21]

2.1.5 Autoformer

The Autoformer[WXW21] is a transformer-based method for times series forecasting that replaces scaled dot product attention with a novel attention function based on auto-correlation,

coupled with trend-seasonality decomposition components in the encoder and decoder blocks. The encoder component is composed of auto-correlation blocks that structurally resemble attention heads, and feed-forward network blocks, each followed by series decomposition, with residual connections. The auto-correlation mechanism of the Autoformer replaces dot product attention with a computation based on autocorreltion. Autocorrelation measures the correlation between a time series and a lagged version of itself. The autocorrelation at lag τ is given by

$$\mathcal{R}(\tau) = \lim_{L\to\infty} \sum_{t=1}^{L} X_t X_{t-\tau}$$

This can be computed using the Fast Fourier Transform, taking the inverse Fourier transform of the power spectral density.

$$R(\tau) = \mathcal{F}_\tau^{-1}\left(\mathcal{F}(X_t)\mathcal{F}^*(X_t)\right)$$

The auto-correlation component of the Autoformer first finds the k lags with highest auto-correlation scores, then normalizes them via softmax transformation. These normalized scores function analogously to attention weights. Then we apply Roll operation, by which lag of magnitude τ_i is applied to sequence V, where beginning part is wrapped around to the end. Then the Auto-Correlation mechanism is the sum of the sequence rolled by lag τ_i and the normalized autocorrelation score $\hat{R}(\tau_i)$. This is analogous to the attention pooling in standard use attention; however, there is a subtle difference. In standard attention, the elements of V are aggregated by weighted sum; in the Autoformer's Auto-Correlation mechanism, the autocorrelation-derived weights are multiplied element-wise to the rolled time series and summed over all chosen lags.[WXW21]

The second novel component of the Autoformer is the use of series decomposition into trend and seasonality components. The trend component is computed using a weighted average with a fixed window size, and the seasonality component is defined as the difference between the original series and its trend component.

$$X_{seasonal}, X_{trend} = \text{SeriesDecomp}(X)$$

where

$$X_{\text{trend}} = \text{AvgPool}(\text{Padding}(X))$$

$$X_{\text{seasonal}} = X - X_{\text{trend}}$$

The Autoformer consists of encoder and decoder components which are similar in structure to those used by the Vanilla Transformer. The encoder component has two components: an autocorrelation component followed by residual connections and series decomposition; and a feed forward network with residual connections followed by series decomposition. With $l = 1, ..., N$ representing the encoder index, we formalize the encoder structure as follows

$$X^{l,1}_{\text{en,seasonal}}, = \text{SeriesDecomp}(\text{Auto-Correlation}(X^{l-1}_{\text{en}}) + X^{l-1}_{\text{en}})$$

$$X^{l,2}_{\text{en,seasonal}}, = \text{SeriesDecomp}(\text{FeedForward}(X^{l,1}_{\text{en,seasonal}}) + X^{l,1}_{\text{en,seasonal}})$$

The decoder component consists of three components similar to the Vanilla Transformer encoder, in addition to a trend-cyclical component. The first component is an Auto-Correlation component, with residual connections followed by series decomposition; the second is an Auto-Correlation component that takes as input the output of the previous component in addition to the output of the encoder, with residual connections and series decomposition; the third component is a feed forward network followed by residual connections and series decomposition. The trend-cyclical component accumulates the trend output of the series decomposition components, and aggregates them as a weighted sum, with parameterized weight matrices learned by the model. For $l = 1, ..., M$ decoder layers, we have

$$X^{l,1}_{\text{de,seasonal}}, X^{l,1}_{\text{de,trend}} = \text{SeriesDecomp}(\text{AutoCorrelation}(X_{\text{de}^{l-1}}) + X^{l-1}_{\text{de}})$$

$$X^{l,2}_{\text{de,seasonal}}, X^{l,2}_{\text{de,trend}} = \text{SeriesDecomp}(\text{AutoCorrelation}(X^{l,1}_{\text{de,seasonal}}, X^{N}_{\text{en}}) + X^{l,1}_{\text{de}})$$

$$X^{l,3}_{\text{de,seasonal}}, X^{l,3}_{\text{de,trend}} = \text{SeriesDecomp}(\text{FeedForward}(X^{l,2}_{\text{de,seasonal}}, X^{N}_{\text{en}}) + X^{l,2}_{\text{de}})$$

$$X^{l}_{\text{de,trend}} = X^{l-1}_{\text{de,trend}} + W_{l,1} * X^{l,1}_{\text{de,trend}} + W_{l,2} * X^{l,2}_{\text{de,trend}} + W_{l,3} * X^{l,3}_{\text{de,trend}}$$

$$X^{l}_{\text{de}} = X^{l,3}_{\text{de,seasonal}}$$

The output is constructed by summing the final trend component and a projection of the final seasonal component into the desired output dimension, where the projection matrix W_s is a parameterized matrix learned by the model.[WXW21]

$$\hat{Y} = W_s X_{\text{de}}^M + T_{\text{de}}^M$$

2.1.6 FEDFormer

The FEDFormer[ZMW22] shares much structurally and conceptually with the Autoformer. Both replace scaled dot product attention with a novel calculation that is based on time series analysis principles. While the Autoformer bases its calculation on auto-correlation, the FEDFormer performs Fourier decomposition and works in the spectral domain. The FEDFormer reduces computational complexity selecting Fourier components in the spectral domain before converting features back into the time domain. The FEDFormer additionally reduces computational complexity by performing attention-like computation in the frequency domain. Finally, the FEDFormer takes up the trend-seasonality decomposition of the Autoformer and replaces it with a more sophisticated mixture of experts decomposition. [ZMW22]

The FEDFormer consists of encoder-decoder structure similar to the Autoformer. Its first component consists of a Frequency Enhanced Block, which decomposes the signal via the Fourier transform, randomly selects a number of Fourier components, transforms the sparse matrix of Fourier components by a parameterized matrix learned by the model, and converts back to the time domain by Inverse Fourier transform. This step is followed by residual connections and mixture of experts decomposition. The second component consists of a feed forward network with residual connections followed by mixture of experts decomposition. The Frequency Enhanced Block can be represented

$$\text{FEB}(q) = \mathcal{F}^{-1}(\text{Padding}(\text{Select}(\mathcal{F}(q)) \odot R)$$

where the select function is a random selection of components of $\mathcal{F}(q)$, and R is a param-

eterized matrix learned by the model. The mixture of experts decomposition method for trend extraction uses moving averages, just as the standard fixed-window moving average. However, the MOE decomposition method computes several moving averages of windows with different sizes, and combines them via softmax and weighting.

$$X^{l,1}_{\text{en,seasonal}}, = \text{MOEDecomp}(\text{FEB}(X^{l-1}_{\text{en}}) + X^{l-1}_{\text{en}})$$

$$X^{l,2}_{\text{en,seasonal}}, = \text{MOEDecomp}(\text{FeedForward}(X^{l,1}_{\text{en,seasonal}}) + X^{l,1}_{\text{en,seasonal}})$$

and $X^{l}_{\text{en}} = X^{l,2}_{\text{en,seasonal}}$ The decoder component is very similar to the Autoformer's decoder. The first component is a frequency enhanced block with MOE decomposition, and the second component is frequency enhanced attention block, which combines the output of the previous component with the output of the encoder block. The frequency-enhanced attention block functions by applying Fourier transform, sampling components randomly, performing dot product attention aggregation using the sparse frequency-domain vectors, then converting back to the time domain by the inverse Fourier transform.[ZMW22]

By working in the frequency domain, the FEDFormer is able to achieve linear memory and time complexity. Additionally, its novel use of methods from Fourier analysis of time series in place of dot product attention allow the model to more efficiently learn features for time series forecasting. The model out-performs the other transformer based methods reviewed here on standard benchmark datasets.

2.1.7 Temporal Fusion Transformer

The Temporal Fusion Transformer[LAL21] is the most recent transformer-based method for time series forecasting reviewed here, and it is unique among the models considered here in many ways. Unlike the methods reviewed above, the Temporal Fusion Transformer does not prioritize memory efficiency. It is similar to the other reviewed transformer methods, however, in its modification of the Vanilla Transformer to enhance local context learning. This is accomplished with LSTM components to extract meaningful local contextual fea-

tures. The Temporal Fusion Transformer has many elements designed to facilitate model interpretation, including feature selection components, gating mechanisms, a modification of multi-head attention that enhances interpretability, and uncertainty quantification via quantile loss.[LAL21]

The Temporal Fusion Transformer allows covariates that are not time-dependent, as well as covariates only known up to the time of prediction and future covariates which may be known into the future at the time of prediction. Each covariate is passed through a variable selection component. [LAL21]

Where other transformer based approaches have used convolution or time-series analysis based computations to enhance learning of local context, the Temporal Fusion Transformer uses an LSTM encoder component to learn local context. The outputs of the LSTM encoder are passed through a gate component with residual connections and layer normalization, then are combined with the output of the static covariate encoder via a gated residual network. The output is then passed through interpretable multi-head attention. [LAL21]

The Temporal Fusion Transformer modifies the standard multi head attention to enable interpretation. In the standard case, the values V are projected a different subspace for each head, and the attention weights therefore are not readily interpretable. The Temporal Fusion Transformer addresses this by using the same value weights W_V across all heads.

$$\text{InterpretableMHA}(Q, K, V) = H_{\text{interpretable}} W_H$$

$$H_{\text{interpretable}} = \frac{1}{H} \sum_{h=1}^{H} \text{Attention}(QW_Q^h, KW_K^h, VW_V)$$

Observe that W_Q and W_K are different projections for each head, but W_V is invariant across heads. This ensures that the model learns attention weights for the same set of input values V. [LAL21]

2.2 MLP-Based Time Series Methods

2.2.1 NBEATS and NBEATSx

NBEATS[OCC19] and its extension NBEATSx[OCM23] are MLP-based methods for time series forecasting that are computationally lightweight compared to transformer-based models, allowing for faster training and inference. Additionally, they outperform most transformer models on standard benchmark datasets, making them suitable choices for many time series forecasting tasks. NBEATS (and NBEATSx) iteratively decomposes the input signal using projection onto basis functions and produces a forecast by iteratively summing forecasts composed of basis functions by residual connections. Its architecture is composed of multiple stacks, each of which is composed of some number of blocks. Each block consists of a fully connected neural network that outputs basis coefficients for its forecast and back-cast. The blocks are connected with residual connections, where the backcast is subtracted from the previous backcast, and the forecast is summed to the previous forecast. The back-cast of block b serves as input to block $b+1$. More formally, each block takes as input the $b-1$ backcast, and the 0 backcast is the model input consisting of some window of the target time series. We denote the backcast at stack s block b $y^{\text{back}}_{s,b}$. The NBEATSx model is extended from the original to take temporal and static covariates X_0. Then block b at stack s consists of two components, a fully-connected neural network, and a linear layer that outputs the basis coefficients θ.[OCC19] [OCM23]

$$h_{s,b} = \text{FCNN}_{s,b}(y^{\text{back}}_{s,b-1}, X_{b-1})$$

$$\theta^{\text{back}}_{s,b}\text{LINEAR}^{\text{back}}(h_{s,b})$$

$$\theta^{\text{for}}_{s,b}\text{LINEAR}^{\text{for}}(h_{s,b})$$

The backcast and forecast predictions at block b and stack s are constructed a set of chosen basis functions and the learned coefficients

$$\hat{y}^{\text{back}}_{s,b} = V^{\text{back}}_{s,b}\theta^{\text{back}}_{s,b}$$

$$\hat{y}^{\text{for}}_{s,b} = V^{\text{for}}_{s,b}\theta^{\text{for}}_{s,b}$$

At each block, the backcast form the previous block is subtracted from the input, simplifying the learning task of each block.

$$\hat{y}^{\text{back}}_{s,b+1} = y^{\text{back}} - \hat{y}^{\text{back}}_{s,b}$$

and

$$\hat{y}^{\text{for}}_{s} = \sum_{b=1}^{B} \hat{y}^{\text{for}}_{s,b}$$

Similarly, the backcast of each stack is the input to the successive stack. This structure is reminiscent of trend-seasonality decomposition in which a trend is modeled and the seasonality component is taken to be the remainder $y_{\text{seasonality}} = y - y_{\text{trend}}$. This is in fact the intuition behind the interpretable configuration proposed with the NBEATSx model. The authors propose using appropriately chosen basis functions for each stack to produce trend and seasonality decomposition and a third component governed by the effect of exogenous variables. For the trend component, a polynomial basis is selected, and the network predicts coefficients of those polynomials. In the seasonality stack, the basis is chosen to be a Fourier basis, and the model learns Fourier coefficients to represent the periodic patterns in the target data. And finally the exogenous variable stack can be thought of as a "time-varying local regression".[OCM23]

2.2.2 NHITS

NHITS improves the NBEATS model by reducing memory requirement and improving accuracy of long-horizon predictions[COO23]. While NBEATS learns components of a time series decomposition using Fourier and polynomial bases, NHITS learns components by interpolating values sampled at multiple frequencies. It does so by multi-rate sampling of the data and a novel form of multi-scale synthesis to produce the forecast and the original granularity. N-HiTS shares the block and stack decomposition structure of NBEATS and

NBEATSx. However, instead of projecting the signal onto basis functions, the signal is sampled and interpolated at different sampling rates, which allows the model to learn patterns at different sampling frequency levels.[COO23]

We choose a sampling rate k_b for each block b and sample the input using a MaxPool operation. That downsampled data is then passed through an MLP layer that outputs forecast and backcast coefficients that are used to synthesize the backcast and forecast at that block. The number of coefficients is dictated by a chosen expressiveness ratio r_b. The expressiveness ratio effectively controls the number of model parameters per output unit of time [COO23].

$$y^p = \text{MaxPool}(y, k_b)$$

Similar to NBEATS and NBEATSx, each block produces a backcast and forecast, where each successive backcast is subtracted from the previous, and the forecasts are summed to produce the final result. However, because of the multi-rate sampling approach of N-HiTS, the components cannot be summed directly, and they need to be reconciled to the frequency of the target output. To do this, the model hierarchically interpolates the value from its components at the required time steps t. The expressiveness ratios are chosen for each block to restore the down-sampled components to the original granularity by the time they are output of the final block.[COO23]

2.3 Gradient Boosted Trees

2.3.1 Regression Trees

Regression trees partition the feature space X into j regions R_j, $j = 1, ..., J$, and assign a constant prediction γ_j for inputs in that region. A regression tree can be formalized as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$$

Where the tree is defined by its parameters $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$. Finding the global solution of partitions R_j and prediction constants γ_j is intractable, so regression trees are fit by a greedy stagewise approach. At the jth stage, we choose R_j, defined by a splitting variable and partition point among values of that variable, that produce a j-node tree that minimizes the loss function. The constants γ_j are chosen as the mean of response values y_i corresponding to $x_i \in R_j$. This approach is a greedy approach in that it does not adjust prior partitions and instead only adds the next nodes minimize the loss function [HTF09].

2.3.2 Boosted Tree Models

Boosted tree models are stagewise additive models comprised of trees. Formally, a boosted tree model can be represented as

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

where each $T(x; \Theta_m)$ is chosen to minimize the loss function in a greedy stagewise fashion. When using squared error loss for boosted regression trees, the mth tree is the one with parameters

$$\hat{\Theta}_m = \text{argmin}_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m) \tag{2.1}$$

$$= \text{argmin}_{\Theta_m} \sum_{i=1}^{N} (y_i - (f_{m-1}(x_i) + T(x_i; \Theta_m))^2 \tag{2.2}$$

$$= \text{argmin}_{\Theta_m} \sum_{i=1}^{N} ((y_i - f_{m-1}(x_i)) - T(x_i; \Theta_m))^2 \tag{2.3}$$

$$= \text{argmin}_{\Theta_m} L((y_i - f_{m-1}(x_i)), T(x_i; \Theta_m))^2 \tag{2.4}$$

Thus each new tree in the boosted regression tree model with squared error loss is equivalent to fitting a regression tree to the residuals of the boosted tree model with $m-1$ trees [HTF09].

2.3.3 Gradient Boosted Tree Models and XGBoost

Gradient boosted trees extend the idea of boosted tree models by choosing each new tree with a method closely related to gradient descent. It is generalizable to any differentiable loss function. In our case, the gradient of the loss function is represented as

$$g_{im} = \left[\frac{\partial}{\partial f(x_i)} L(y_i, f(x_i)\right]$$

Then our update step is

$$f_m = f_{m-1} - \rho_m g_m$$

where ρ_m is the step size at step m. Because our model is constrained to be an ensemble of regression trees, we are not free to use g_m exactly. Instead, we find the tree $T(x; \tilde{\Theta})$ that best approximates g_m. That is, at each step, we choose $f_m = T(X; \tilde{\Theta}_m)$ with

$$\tilde{\Theta}_m = \text{argmin}_\Theta \sum_{i=1}^{N} (-g_{im} - T(x_i; \Theta))^2$$

We recognize this as a standard regression tree problem. Further, when we use squared error loss, the negative gradient is equivalent to the residual

$$-\frac{\partial}{\partial f(x_i)} \frac{1}{2}(y_i - f(x_i))^2 = y_i - f(x_i) \tag{2.5}$$

XGBoost is a tree boosting algorithm that is efficient and scalable[CG16]. It uses a loss function with a regularization term

$$\mathcal{L} = \sum_{i=1}^{n} L(y_i, f_m(x_i)) + \sum_{i=1}^{m} \left(\lambda_1 T + \frac{1}{2}\lambda_2 \|\gamma\|^2\right)$$

where T represents the number of nodes in the $m+1$th tree and $\|\gamma\|^2$ is the squared norm of the vector of value γ_i. The regularization parameters λ_1 and λ_2 must be chosen separately from the optimization process. Instead of computing the negative gradient of $\mathcal{L}$ exactly, it is approximated by its second-order Taylor approximation, given by

$$\mathcal{L} \approx \sum_{i=1}^{n} \left(L(y_i, f_m(x_i)) + g_i f_m(x_i) + \frac{1}{2} h_i f_m^2(x_i)\right) + \left(\lambda_1 T + \frac{1}{2}\lambda_2 \|\gamma\|^2\right)$$

with $g_i = \frac{\partial}{\partial f_m(x_i)} L(y_i, f_m(x_i))$ and $h_i = \frac{\partial^2}{\partial^2 f_m(x_i)} L(y_i, f_m(x_i))$ The algorithm proceeds as in the gradient boosted tree model, choosing the variable and split that reduces $\mathcal{L}$ the most. [CG16]

Another feature that can improve the performance of XGBoost is the choice to use an approximate split finding approach alternative to the exact split finding procedure. In the exact approach, at each iteration of the greedy algorithm, each datapoint is considered as a potential split point, and the split point that offers the greatest loss reduction is chosen. In the approximate method, the data is partitioned into bins approximating quantiles, and the algorithm considers splits at each pre-determined candidate split point. This can reduce the computational burden greatly, and it solves the problems caused by datasets that are too large to fit in memory at one time and the difficulty of performing gradient tree boosting on distributed systems. The approximate split finding approach has two options: to determine the approximate candidate splits "globally" once before the algorithm begins; or to re-determine the candidate splits of a region if that region becomes split[CG16].

In addition to the Tikhonov regularization terms, XGBoost allows regularization via a shrinkage parameter and column subsampling. The shrinkage parameter is analogous to the step size parameter in gradient descent, as it scales the contribution of each new tree[CG16]. Column subsampling involves randomly selecting a portion of available features at each boosting round, where the number of features selected is controlled by a parameter. [CG16]

2.4 Conformal Inference

Conformal inference is method of generating uncertainty sets or intervals for predictive models that are guaranteed to have a pre-specified level of uncertainty. While there are many methods available to statisticians for uncertainty quantification, conformal inference is attractive because it requires no assumptions about the model and minimal assumptions about the distribution generating the data. Further, many conformal inference algorithms are rela-

tively computationally inexpensive, making them simpler and faster alternatives to methods like the bootstrap. The single assumption required by standard conformal inference procedures is that the data be exchangeable. This is a slightly weaker assumption than i.i.d., and it means that the joint distribution of the data does not change if the order of the data is altered.

This assumption of exchangeability does not hold for time series data because of the inherent temporal dependence. There has been recent innovative work that extends conformal inference to the time series setting, and we review some of the current approaches below. We review the adaptive conformal inference (ACI) approach, and a recent extension that allows it to be fit in a parameter-free way.

Conformal prediction is not a single algorithm but rather a constellation of methods that share the following basic procedure [AB23], outlined below

1. Identify a heuristic notion of uncertainty with a pre-trained model
2. Define a score function $s(X, Y) \in R$ such that larger score represents worse agreement between x and y
3. Computer the $1 - \alpha$ quantile $\hat{Q}(1 - \alpha)$ of the calibration scores
4. Use this quantile to form the prediction set for new examples:

$$C(X_{\text{new}}) = \{y : s(X_{\text{new}}, y) \leq \hat{Q}(1 - \alpha)\}$$

When the data is exchangeable, we are guaranteed that

$$1 - \alpha \leq P(Y_{\text{new}} \in C(X_{\text{new}})) \leq \frac{1}{n+1} + 1 - \alpha$$

where n represents the size of the set of observations for which calibration scores have been generated [AB21]. In[LGR18], the upper bound is proven to hold when the distribution of $S(x, y)$ is continuous, but Angelopoulos and Bates claim that this result holds even when that condition is not met. [AB21]

The intuition behind this result is stated most clearly in section 2 of [LGR18], where it is proven for the case of conformal inference for regression. When the data is exchangeable, the calibration score s_{new} has equal probability of falling between any two of the ordered conformity scores generated for observations $1...n$, $s_1, ..., s_n$. In other words, the rank of s_{new} among the order statistics $s_{(1)}, ..., s_{(n)}$ is uniformly distributed over the set $\{1, ..., n, n+1\}$ and therefore has probability $\frac{1}{n+1}$ of being in the interval $[s_{i-1}, s_i]$ $i = 1...n+1$. We do not reproduce the full proof here, but this insight should provide an intuitive understanding of the connection between exchangeability and the coverage guarantee stated above. Thus when the data is not exchangeable, we are not guaranteed this bound on the coverage of a prediction set. It is for this reason that the authors in the papers reviewed below have developed new procedures to produce conformal prediction sets in settings without exchangeability and new theory for coverage guarantees in these settings.

When the underlying distribution generating the data is not stationary, the exchangeability assumption required for the coverage guarantees of standard conformal inference procedures is violated. Candes and Gibbs, in [GC21], propose procedures to produce valid conformal prediction sets in this setting.

To motivate this problem, define the realized miscoverage rate of a prediction set

$$M_t(\alpha) = P(S_t(X_t, Y_t) > \hat{Q}_t(1-\alpha))$$

Where $S_t(X_t, Y_t)$ is the conformity score and $\hat{Q}_t$ is the corresponding quantile function. The miscoverage rate can be understood as the probability that the prediction set generated using $\hat{Q}(1-\alpha)$ does not contain the true value. With a valid conformal prediction procedure, we expect $M_t(\alpha) = \alpha$, but this cannot be expected to hold when the distribution generating the data is not stationary.Further, Gibbs and Candes argue that there exists an $\alpha_t^* := \sup\{\beta \in [0,1] : M_t(\beta) \leq \alpha\}$ which will give us $M_t(\alpha^*) = \alpha$. This is the basis for the proposed algorithm.

The Adaptive Conformal Inference procedure involves updating α to increase or decrease

the size of $\hat{C}_t(\alpha)$ based on past under or over coverage. This is done by the following simple online update

$$\alpha_{t+1} = \alpha_t + \gamma(\alpha - \text{err}_t)$$

where γ is a fixed step-size parameter and

$$\text{err}_t = \begin{cases} 1 & Y_t \notin \hat{C}_t \\ 0 & \text{otherwise} \end{cases}$$

The paper indicates that γ should be chosen to be larger when the distribution shift is large, but gives no specific theory about choosing the optimal step size.

Gibbs and Candes prove an asymptotic coverage guarantee, which does not rely on any assumption of the data-generating distribution. This proposition ensures that the ACI algorithm attains good coverage over long time periods, in particular that as $t \to \infty$ $M_t(\alpha) \to \alpha$ The ACI approach is extenced by in [ZFG22]. Zaffran et. al. propose a parameter-free algorithm that uses online expert aggregation to select the optimal value of γ at each step.

CHAPTER 3

Methods

3.1 Experiments and Methodology

We compare 4 models applied to the day-ahead EPF problem, using PJM market data. The models compared are an XGBoost model with engineered features, NBEATSx, NHiTS, and Temporal Fusion Transformer.

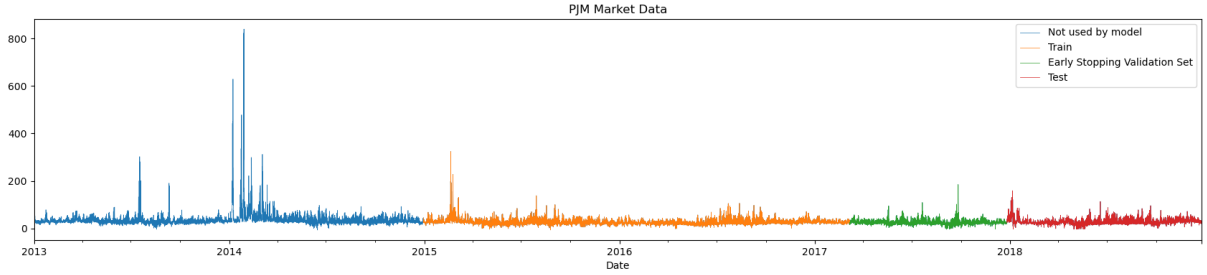

Figure 3.1: Train Val Test Split

While we followed the training and testing procedures in [LMD21] and [OCM23] as closely as possible, modifications were made to simplify the experiments and reduce the computational burden. Therefore the accuracy results we obtain are not directly comparable to the published results in [LMD21] and [OCM23]. The most notable modification is that instead of using a prediction set of length 2 years, we reduced it to 1 year to reduce the compute time while remaining within the recommendations of [LMD21]; additionally, we only test the models on the PJM market data, instead of comparing multiple markets. This gives us less insight into the models' abilities to handle different levels of volatility. We have produced predictions for the NBEATSx and LEAR models on the same one year of data

for comparison to the other models studied here. Finally, as detailed below, we explore a smaller parameter space or fit models using fewer hyperparameter tuning iterations than the published versions.

We use roughly the same model fitting and prediction strategy for each model. Each model is trained on 3 years of data. [LMD21] treat length of calibration window as a hyperparameter, and demonstrate that it does influence the accuracy of the model. Generally, models trained on longer calibration windows are influenced by older market dynamics that may no longer be relevant to current dynamics, while models trained on shorter periods suffer from a smaller training sample. Early stopping is used, with a early stopping validation set of 42 weeks. In the XGBoost model, the validation set is the 42 weeks prior to the test period, while in the deep learning models, the validation set is randomly sampled from the training set.

A key component of our approach to the models is daily calibration, following [LMD21]. Because the market dynamics are assumed to change, the model fit on our test data may not adequately handle future market dynamics. The hyperparamters selected in the tuning phase remain the same, while the model is refit using those hyperparameters before predicting each new day.

The NBEATSx, NHiTS and TFT models are implmented using the Nixtla library in Python. The hyperparameter tuning is also done using Nixtla, and Ray libraries in Python. The XGBoost hyperparameter training is done using the Python library Optuna.

We use data provided by the EPFToolbox[LMD21]. The EPFToolbox provides data from five energy markets for day-ahead price forecasting model benchamarking. For each model, two time-series covariates are provided that have been found to be helpful in forecasting the market's prices. In this project, we use the Pennsylvania-New Jersey-Maryland (PJM) dataset. The target variable is the zonal electricity price in Commonwealth Edison. The two covariates are day-ahead system load forecast and day-ahead COMED zonal load[LMD21]. The PJM electricity prices exhibit frequent spikes in comparison to other markets, making

it a difficult market to forecast accurately. The dataset covers 12/27/2016 to 12/24/2018. We predict the last year of data. The model is trained on the three years of data prior to the test period.

We evaluate prediction accuracy using root mean squared error (RMSE) and mean absolute error(MAE). RMSE is more sensitive to large errors, while MAE is not.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Implementation and Training

We implement a model using XGBoost. Data for autoregressive problems can be represented in a tabular format by simply adding lag features as new columns. We train three separate models, one containing only one day of lags of the target variable; one with lags from days $d-1, d-2, d-3, d-7$ for the target variable and the two covariates; and one with lags $d-1, d-2, d-3, d-7$ in addition to a number of engineered features. The engineered features include rolling mean, rolling median, rolling variance, and exponentially weighted average over windows of size 7 days and 112 days applied to the target variable and the two covariates. Given the special nature of this problem, special care was taken to ensure that no same-day data leakage was introduced with the lags or the engineered features using the target variable. For each of the 24 hours in a day, we start counting the lags at the previous midnight, so that no row contains data from the same day, and same-day values were excluded from the rolling window features calculated with the target variable.

We compute prediction intervals using the ACI method. We select $\gamma = 0.005$, but did not compare the performance of the algorithm with multiple γ values. While there are many conformal prediction methods for time series data that appear to perform better in terms of interval sharpness and validity, such as EnbPI [XX23a], SCPI [XX23b], and the extension of

ACI by Zaffran [ZFG22], there were considerations that ruled them out. EnbPI requires the use of bootstrap estimators, which is impractical for our models which are time consuming to train and are re-trained for each prediction day. Zaffran's approach does not require bootstrap estimation, but implementation of the online aggregation approach was out of scope for this project. Additionally, while the standard variant of the SCPI algorithm does not require bootstrap estimation, the SCPI approach for multi-step predictions proposed in [XX23b] does rely on bootstrap estimators to compute the joint density of the error terms of the forecast horizon. We show below that the standard ACI algorithm produces reasonable prediction intervals with realized coverage close to the desired coverage level.

Table 3.1: Hyperparameter search space for XGBoost models

Hyperparameter	**Considered Values**
Max tree depth	$[3, \ldots, 10]$
η	$[1e-8, 1]$ (log scale)
γ	$[1e-8, 1]$
Grow Policy	$\{\text{Depthwise}, \text{lossguide}\}$
Subsample	$[0.5, 1.0]$
Column Sample by	$[0.5, 1.0]$
Tree	
α	$[1e-8, 1]$
λ	$[1e-8, 1]$

Results

Table 3.5 displays the rMSE and MAE scores for each model tested. We find that NBEATSx and NHiTS perform the best, outperforming all models including the LEAR benchmark. The XGBoost models do not perform better than the LEAR benchmark, and outperform only the TFT model, which performs worst. The simplest XGBoost model performs best with

Table 3.2: Hyperparameter search space for NBEATSx model with optimal values

Hyperparameter	**Considered Values**	**Optimal Value**
Activation function	{SoftPlus, SeLU, PReLU, Sigmoid, ReLU, Tanh, LeakyReLU}	LeakyReLU
FCNN hidden neurons on each layer of a block.	$\{50, 100, 150, \ldots, 450, 500\}$	450
Number of Fourier bases in seasonality component	$\{1, 2\}$	2
Degree of polynomial in trend component	$\{2, 3, 4\}$	3
Dropout probability	$[0, 1]$	0.338
Scaler Type	standard, minmax, robust, identity	robust

regard to rMSE, but the second and third most complex XGBoost models perform better with respect to MAE.

In figure 3.2 we plot each models predictions over two weeks. The first week is one where most of the models performed poorly, and the second is the week on which all models performed best.

In tables 3.2, 3.3 and 3.4, we display the hyperparameter space for each model and the hyperparameter values selected in the hyperparameter tuning process.

Finally, we considered model combinations. It is often the case that models that are diverse in some respect yield better results when combined by averaging. Table 3.6 shows the best combinations computed by arithmetic mean of each of 2 to 6 number of models. We find that each of these combinations yield drastic improvements in accuracy, outperforming any of the best single models we studied.

Table 3.3: Hyperparameter search space for NHiTS model with optimal values

Hyperparameter	**Considered Values**	**Optimal Value**
Activation function	{SoftPlus, SeLU, PReLU, Sigmoid, ReLU, Tanh, LeakyReLU}	ReLU
FCNN hidden neurons on each layer of a block.	512	512
Downsample Ratios	$[4, 2, 1]$, $[168, 24, 1]$, $[24, 12, 1]$, $[40, 20, 1]$, $[64, 8, 1]$	$[168, 24, 1]$
Pooling Kernel Size	$[2, 2, 2]$, $[4, 4, 4]$, $[8, 8, 8]$, $[8, 4, 1]$, $[16, 8, 1]$	$[8, 4, 1]$
Pooling Mode	{MaxPool1d, AvgPool1d}	MaxPool1d
Dropout probability	$[0, 1]$	0.513
Scaler Type	standard, minmax, robust, identity	standard

We use Adaptive Conformal Inference to produce 95% prediction intervals. The results are shown in Figure 3.3. We plot the daily average coverage and the interval width below each prediction interval plot. We see that the prediction intervals attain close to the desired coverage, and that the width of prediction intervals is wider in periods of higher volatility when the predictions tend to be less accurate.

Table 3.4: Hyperparameter search space for TFT model with optimal values

Hyperparameter	**Considered Values**	**Optimal Value**
Hidden size	$\{8, 20, 40, 80, 160\}$	160
Number of Attention Heads	$\{1, 4\}$	1
Dropout	$[0, 1]$	0.615
Scaler Type	standard, minmax, robust, identity	minmax

Table 3.5: Model Test Error

	XGB1	XGB2	XGB3	NBEATSx	NHiTS	TFT	LEAR
MAE	4.13	4.00	4.02	3.30	3.51	4.71	3.65
RMSE	6.67	6.94	6.75	5.58	5.74	7.33	6.15

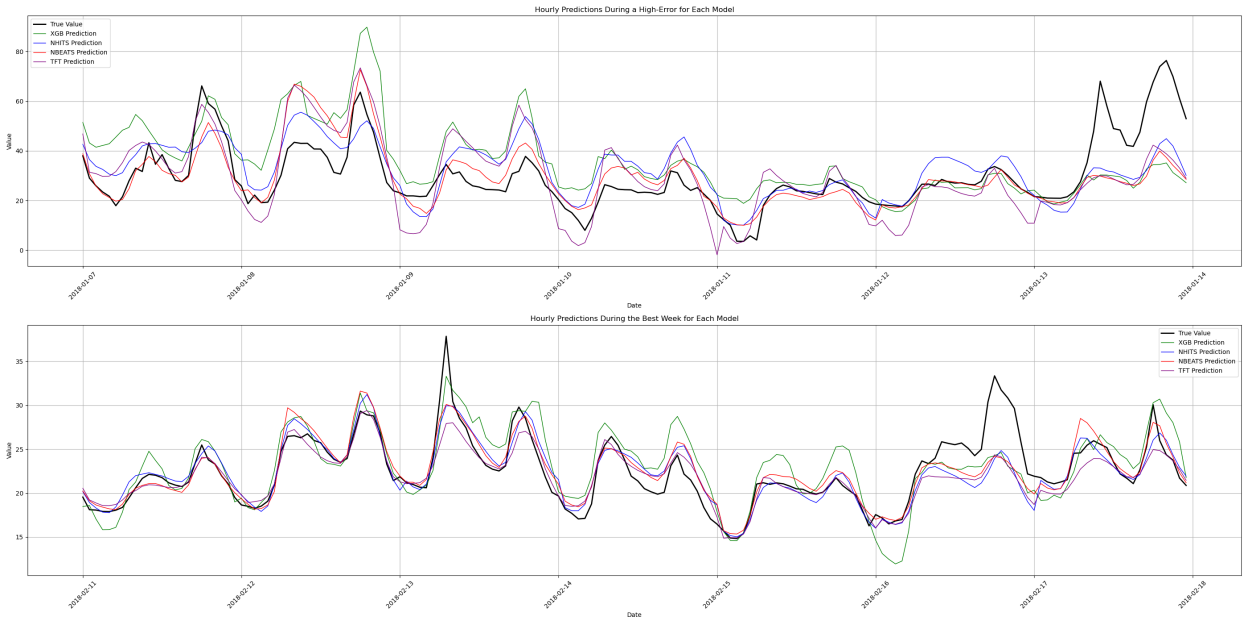

Figure 3.2: A low-error week and high-error week for each model

Table 3.6: Best Model Combinations for rMSE and MAE. XGB1 represents the XGBoost model with only $d-1$ lags on the target variable; XGB2 is the XGBoost model with all lags; XGB3 is the XGBoost model with all lags and engineered features

Number of Models	**Best Combination**	**rMSE**	**MAE**
2	XGB 1, NBEATSx	5.234	3.244
3	XGB 1, NBEATSx, NHiTS	**5.156**	**3.177**
4	XGB 1, XGB 3, NBEATSx, NHiTS	5.317	3.238
5	XGB 1, XGB 2, TFT, NBEATSx, NHiTS	5.412	3.293
6	XGB 1, XGB 2, XGB 3, TFT, NBEATSx, NHiTS	5.514	3.331

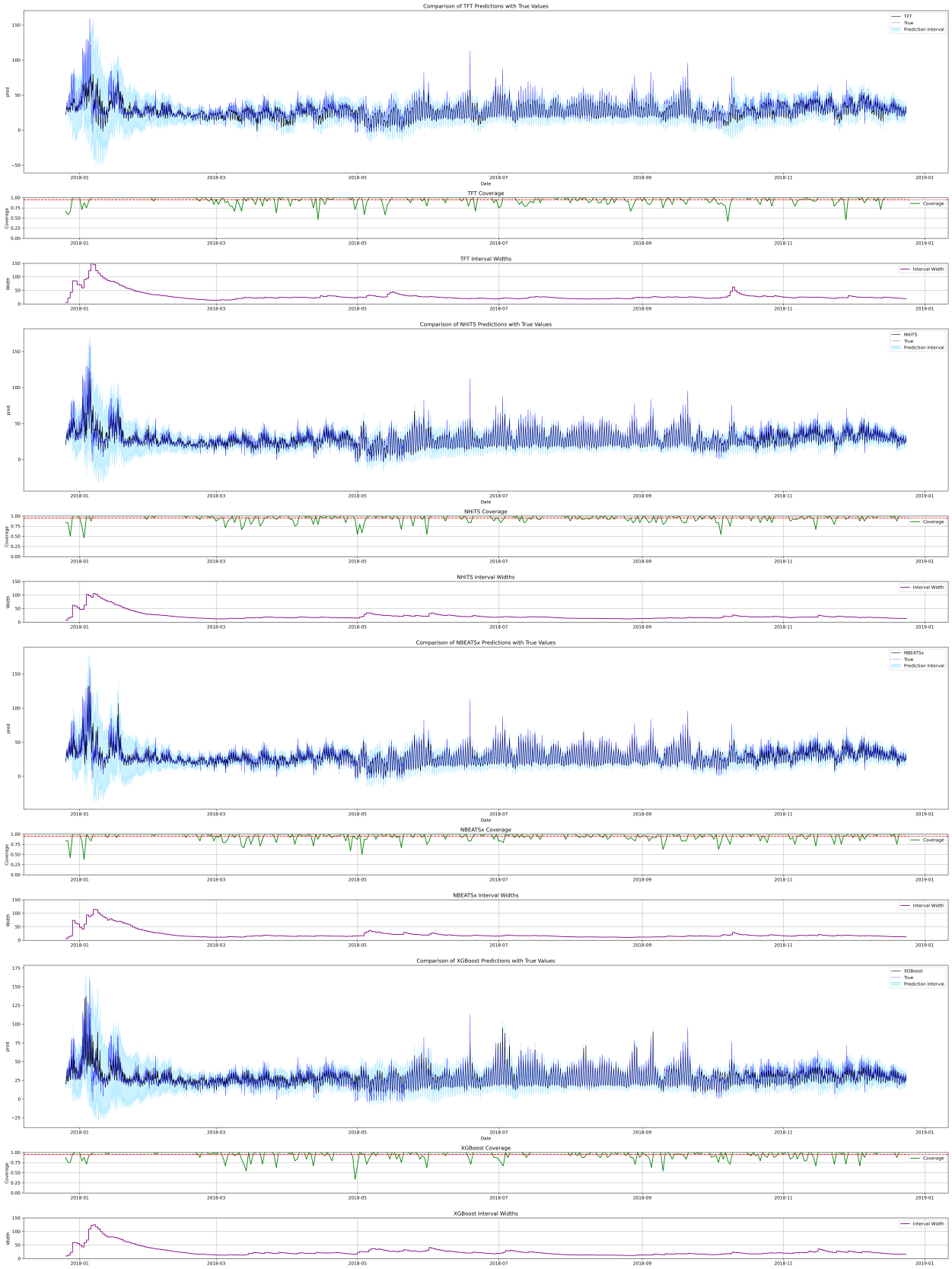

Figure 3.3: Model predictions with ACI prediction intervals

CHAPTER 4

Conclusion

We have tested four approaches to day-ahead electricity price forecasting. We find that NBEATSx and NHiTS are strong performers when applied to this problem. The XGBoost models did not perform well in comparison to NBEATSx, NHiTS and the LEAR benchmark. We found anecdotally that increasing the maximum depth of trees greater than 10 slightly improved performance at the expense of much greater computation time. This increase in performance did not make the XGBoost model perform better than the LEAR benchmark. It is reasonable to expect that further experimentation with feature engineering could improve the performance of the model. Additionally, while we adopted the train-test split pattern used in the EPF benchmarks, it may be beneficial to employ some form of cross-validation for more robust model fitting. Finally we observe a key difference between the LEAR model and XGBoost models. While both models are recalibrated daily, in the case of LEAR, the regularization hyperparameter is refit in each recalibration step. Thus the model is free to select different input variables for each day. In our XGBoost implementation, the hyperparameters, including the regularization parameters, are fit once prior to the prediction phase. The recalibration phase for our XGBoost models entails instantiating and fitting a new model with unchanged hyperparameters. While doing a full hyperparameter search for each recalibration step would be prohibitively time consuming, one fruitful vein of future research may be to consider a method for refitting hyperparameters either over a reduced space or with simplified models. In fact, the LEAR approach finds the lasso hyperparameter using least angle regression during the daily recalibration step. It is possible that improving the recalibration process could improve the performance of XGBoost models for this problem.

We failed to demonstrate the effectiveness of the Temporal Fusion Transformer to this problem. It is possible that with more hyperparameter tuning, we could improve the model's performance. However, Zeng et. al. have recently called into question the effectiveness of transformer models for time series forecasting [ZCZ23]. While all of the transformer models reviewed here reported state of the art performance on a number of benchmark datasets, Zeng et. al. demonstrate they can be outperformed by rather simple models on these same benchmarks. The Temporal Fusion Transformer is not included in Zeng et. al.'s study. However, the TFT is included in Nixtla's large benchmark of time series models [GM23]. They find that LGBM performs best on hourly data, with NHiTS performing second best on hourly data of the approaches studied. In their study, TFT did not out-perform NHiTS on any data frequency. While the Temporal Fusion Transformer has shown to be a very powerful model in time series forecasting for many problems, we are not able to conclude that it is among the best models to puruse for day-ahead electricity price forecasting.

NBEATSx has been demonstrated by [OCM23] to be a top performer in day-ahead electricity price forecasting, in a study that is more thorough than the one conducted here. Our findings confirm their results. We also demonstrate that NHiTS appears to perform nearly as well. Given that NHiTS is reported to perform well with longer time horizons, one possible area of future research would be to evaluate potential benefits of lengthening the prediction horizon and reducing the model recalibration frequency.

REFERENCES

[AB21] Anastasios N Angelopoulos and Stephen Bates. "A gentle introduction to conformal prediction and distribution-free uncertainty quantification." *arXiv preprint arXiv:2107.07511*, 2021.

[AB23] Anastasios N Angelopoulos, Stephen Bates, et al. "Conformal prediction: A gentle introduction." *Foundations and Trends® in Machine Learning*, **16**(4):494–591, 2023.

[BZ24] Jonathan Berrisch and Florian Ziel. "Multivariate probabilistic CRPS learning with an application to day-ahead electricity prices." *International Journal of Forecasting*, 2024.

[CG16] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[COO23] Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. "Nhits: Neural hierarchical interpolation for time series forecasting." In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 6989–6997, 2023.

[GC21] Isaac Gibbs and Emmanuel Candes. "Adaptive conformal inference under distribution shift." *Advances in Neural Information Processing Systems*, **34**:1660–1672, 2021.

[GM23] Azul Garza and Max Mergenthaler-Canseco. "TimeGPT-1." *arXiv preprint arXiv:2310.03589*, 2023.

[HTF09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[KZ21] Christopher Kath and Florian Ziel. "Conformal prediction interval estimation and applications to day-ahead and intraday power markets." *International Journal of Forecasting*, **37**(2):777–799, 2021.

[LAL21] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. "Temporal fusion transformers for interpretable multi-horizon time series forecasting." *International Journal of Forecasting*, **37**(4):1748–1764, 2021.

[LGR18] Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J Tibshirani, and Larry Wasserman. "Distribution-free predictive inference for regression." *Journal of the American Statistical Association*, **113**(523):1094–1111, 2018.

[LJX19] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting." *Advances in neural information processing systems*, **32**, 2019.

[LMD21] Jesus Lago, Grzegorz Marcjasz, Bart De Schutter, and Rafał Weron. "Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark." *Applied Energy*, **293**:116983, 2021.

[MNW23] Grzegorz Marcjasz, Michał Narajewski, Rafał Weron, and Florian Ziel. "Distributional neural networks for electricity price forecasting." *Energy Economics*, **125**:106843, 2023.

[MUW22] Katarzyna Maciejowska, Bartosz Uniejewski, and Rafał Weron. "Forecasting electricity prices." *arXiv preprint arXiv:2204.11735*, 2022.

[OCC19] Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting." *arXiv preprint arXiv:1905.10437*, 2019.

[OCM23] Kin G Olivares, Cristian Challu, Grzegorz Marcjasz, Rafał Weron, and Artur Dubrawski. "Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with NBEATSx." *International Journal of Forecasting*, **39**(2):884–900, 2023.

[UW21] Bartosz Uniejewski and Rafał Weron. "Regularized quantile regression averaging for probabilistic electricity price forecasting." *Energy Economics*, **95**:105121, 2021.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." *Advances in neural information processing systems*, **30**, 2017.

[WXW21] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting." *Advances in neural information processing systems*, **34**:22419–22430, 2021.

[WZZ22] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. "Transformers in time series: A survey." *arXiv preprint arXiv:2202.07125*, 2022.

[XX23a] Chen Xu and Yao Xie. "Conformal prediction for time series." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[XX23b] Chen Xu and Yao Xie. "Sequential predictive conformal inference for time series." In *International Conference on Machine Learning*, pp. 38707–38727. PMLR, 2023.

[ZCZ23] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. "Are transformers effective for time series forecasting?" In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 11121–11128, 2023.

[ZFG22] Margaux Zaffran, Olivier Féron, Yannig Goude, Julie Josse, and Aymeric Dieuleveut. "Adaptive conformal predictions for time series." In *International Conference on Machine Learning*, pp. 25834–25866. PMLR, 2022.

[ZLL21] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. "Dive into deep learning." *arXiv preprint arXiv:2106.11342*, 2021.

[ZMW22] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting." In *International conference on machine learning*, pp. 27268–27286. PMLR, 2022.

[ZZP21] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. "Informer: Beyond efficient transformer for long sequence time-series forecasting." In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11106–11115, 2021.