CONFORMAL INFERENCE FOR ENERGY PRICE FORECASTING

Frederik V. Svane (s224766)
Marcus P. L. Christoffersen (s224750)

ABSTRACT

The abstract should contain about 100 to 150 words summarizing the problem, approach, and key findings.

1. INTRODUCTION

Energy price forecasting, like other investment domains, benefits immensely from uncertainty estimates for proper risk management. Conformal prediction is able to provide these prediction intervals, with guaranteed coverage. Setting a miscoverage rate of $\alpha = 0.1$ means we expect 90% of true values to fall within our intervals. However, standard conformal methods assume exchangeable data, which is violated in time series, where observations are sequentially dependent/inexchangeable.

Extensions to standard conformal inference, such as Adaptive Conformal Inference (ACI) and Online Gradient Descent (OGD), have been developed to address this problem of assumed exchangeability, by dynamically adjusting intervals based on binary coverage feedback. Error-Quantified Conformal Inference (ECI) [1] extends these methods further, by incorporating the *magnitude* of prediction errors, instead of mere binary miss/cover feedback.

In this project, we apply these conformal methods to English electricity spot price forecasting using 2022 hourly data with weather and consumption features. We implement two forecasting models, SARIMAX and LSTM, and compare OGD and ECI for constructing prediction intervals. This allows us to explore how online conformal prediction performs on real energy data from the 2022 energy crisis, where the market experienced increased volatility and distribution shifts.

2. METHODS

2.1. Conformal Prediction

Conformal prediction constructs prediction intervals by comparing new observations to a calibration set. Given a point prediction $\hat{y}_t$ and a true value y_t, we define a nonconformity score $s_t = |y_t - \hat{y}_t|$ measuring how "unusual" the observation is. A prediction interval is then constructed as $[\hat{y}_t - q_t, \hat{y}_t + q_t]$, where q_t is a threshold chosen such that the interval covers the true value with probability $1 - \alpha$.

Standard conformal prediction assumes exchangeability: the joint distribution of observations is invariant to permutation. This assumption fails for time series data, where observations are chronologically dependent (ie. the next spot price depends on the current spot price).

Additionally, the underlying distribution may shift over time, meaning the statistical properties of prices in summer differ from those during winter, and likewise for crisis periods compared to stable periods. So a calibration set from the past may not represent future conditions, causing coverage guarantees to break down.

2.2. Online Conformal Methods

Online conformal methods address non-exchangeability by updating the threshold q_t dynamically based on observed coverage. The Online Gradient Descent (OGD) method uses the update rule

$$q_{t+1} = q_t + \eta \cdot (\text{err}_t - \alpha) \tag{1}$$

where η is a learning rate and the binary variable $\text{err}_t \in \{0, 1\}$ indicates whether the true value fell outside the interval. When coverage is too low ($\text{err}_t = 1$ more often than α), the threshold increases, widening intervals. When coverage is too high, the threshold decreases, tightening intervals.

Error-Quantified Conformal Inference (ECI) extends OGD by incorporating the magnitude of prediction errors. The update rule becomes

$$q_{t+1} = q_t + \eta \cdot [(\text{err}_t - \alpha) + (s_t - q_t) \cdot f'(s_t - q_t)] \tag{2}$$

where s_t is the nonconformity score and f is a shaping function, such as a sigmoid or a Gaussian kernel. As such, the additional term $(s_t - q_t) \cdot f'(s_t - q_t)$ adjusts the update based on how far the score was from the threshold. The choice of shaping function comes with considerations: the sigmoid saturates for large deviations, giving extreme misses similar corrections to moderate ones. The Gaussian kernel goes further, actively dampening corrections for extreme deviations, treating them as potential outliers that should have less influence on the threshold.

2.3. Data

We use hourly electricity market data from England spanning February 1, 2022 to February 1, 2023, comprising 8,760 observations. The target variable is the day-ahead spot price (GBP/MWh), while the raw features include wind power forecasts from a wind farm near London (MW), temperature forecasts (°C), historical normal temperatures (°C), and timestamps.

This period coincides with the European energy crisis, characterized by significant price volatility and distribution shifts. These conditions stress-test conformal methods designed for non-stationary environments.

2.3.1. Feature engineering

With only a single year of data, capturing seasonal patterns directly is challenging. To address this, we engineer additional features from the available measurements.

From the timestamp, we derive hour of day (1–24), day of year (1–365), week number (1–52), and month (1–12). To represent annual seasonality without discontinuities, we encode day of year cyclically:

$$\text{day\_sin} = \sin\left(\frac{2\pi \cdot \text{day}}{365}\right), \quad \text{day\_cos} = \cos\left(\frac{2\pi \cdot \text{day}}{365}\right)$$

We also construct features based on energy market dynamics. Following WHO guidelines that recommend a minimum indoor temperature of 18°C [2], we define heating demand as $\max(0, 18 - \text{temperature forecast})$. We then multiply this by consumption forecasts, based on the hypothesis that cold weather combined with high demand creates price pressure.

Additionally, we compute temperature deviation from historical norms, based on the intuition that unusual weather may drive price volatility.

Finally, we include lagged spot price $(t-1)$ as a feature, reflecting the realistic constraint that only past prices are observable at prediction time.

This yields 14 input features in total. The data is split 80/20 into training and test sets, preserving temporal order to prevent information leakage.

2.4. Forecasting Models

We use two forecasting models to generate point predictions, representing deep learning and classical time series approaches.

2.4.1. Deep Learning Model

Sequential data like this calls for a recurrent architecture. As covered in the lectures, standard feedforward networks treat each input independently, failing to capture positional relationships within a sequence. They also require fixed input dimensions, making them unsuitable for variable-length sequences. Recurrent neural networks (RNNs) address both issues by processing inputs sequentially and maintaining a hidden state that carries information across time steps.

However, traditional RNNs suffer from vanishing or exploding gradients. During backpropagation, gradients in early layers are computed via the chain rule, resulting in a product across all time steps:

$$\frac{\partial \mathcal{L}}{\partial h_0} = \prod_{t=1}^{T} \frac{\partial h_t}{\partial h_{t-1}} \cdot \frac{\partial \mathcal{L}}{\partial h_T}$$

For long sequences, this product either explodes or vanishes. If the intermediate gradients are consistently greater than one, the product blows up. If they are consistently smaller than one, it collapses to zero.

LSTMs address this through a gating mechanism, as we saw in lecture. The cell state acts as a highway that allows gradients to flow across many time steps with minimal transformation. The forget gate controls what information to discard, the input gate controls what new information to store, and the output gate controls what to expose to the next layer. These gates use sigmoid activations to produce values in $[0, 1]$, enabling smooth gradient flow during training.

We considered adding attention mechanisms or moving to a full Transformer architecture, but opted against this for practical reasons. Both require substantially more data to train effectively, and our dataset spans only a single year. They also introduce computational overhead that slows down iteration during experimentation. Given these constraints, a standard LSTM provides a reasonable balance between expressiveness and trainability.

Our implementation uses PyTorch and consists of two stacked LSTM layers with 64 hidden units each, followed by a linear output layer. The network processes sequences of 48 hourly observations (two days), with only the final hidden state passed to the output layer to produce a point prediction for the next hour. The implementation can be found in `models.py` in the linked GitHub repository.

We tune sequence length and number of training epochs via grid search on DTU's HPC cluster. For sequence length, we tested windows ranging from 1 day (24 hours) to 14 days (336 hours), with 48 hours (two days) performing best. The sequence length search is implemented in `seq_len_search.py`, and the epoch search in `LSTM_model_cross_val_train.ipynb`.

Standard k-fold cross-validation is inappropriate for time series, as it allows future information to leak into training folds. Instead, we use chronological cross-validation with an expanding window: train on data up to time t, validate on a fixed horizon ahead, then advance t and repeat.

2.4.2. Classical time series model

3. EXPERIMENTS & RESULTS

3.1. Experimental Setup

3.2. Results

3.3. Analysis

4. CONCLUSION

5. REFERENCES

[1] Junxi Wu, Dongjian Hu, Yajie Bao, Shu-Tao Xia, and Changliang Zou, "Error-quantified conformal inference for time series," 2025.

[2] World Health Organization, *WHO Housing and Health Guidelines*, World Health Organization, Geneva, 2018.

DECLARATION OF USE OF GENERATIVE AI

This declaration **must** be filled out and included as the **final page** of the document. The questions apply to all parts of the work, including research, project writing, and coding.

- I/we have used generative AI tools: yes

If you answered *yes*, please complete the following sections. List the generative AI tools you have used:

- Claude by Anthropic AI
- Claude Code by Anthropic AI

Describe how the tools were used:

What did you use the tool(s) for?

We used the named AI tools for brainstorming- and evaluating ideas, proofreading and correcting parts of the text in the report, bugfixing and generating non-essential, utility code, such as plotting data with matplotlib.

At what stage(s) of the process did you use the tool(s)?

We used the named AI tools during the research-, implementation and proofreading stages of the process.

How did you use or incorporate the generated output?

Some generated output we simply read and responded to, in order to guide our thinking, in a similar way to what we would have done to a TA or professor. Some coding output, such as matplotlib plotting code, we ran directly. And grammatical corrections to sections in our report, we included in the report.