VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
DEPARTMENT OF INFORMATICS

Bachelor’s thesis:

Nordpool electricity price forecasting using artificial intelligence algorithms

Nordpool elektros kainų prognozavimas naudojant dirbtinio intelekto algoritmus

Done by 4th year, 2nd group student:

Rapolas Vanagas

Project supervisor:

Dr. Aistis Raudys

Vilnius
2024

Contents

Summary

The objective of this bachelor's project is to identify the most optimal AI algorithms for predicting Nord Pool electricity prices and to implement techniques that further enhance prediction accuracy. The techniques include extracting relevant features, hyperparameter tuning, and data splits. The models are then trained with the Nord Pool dataset, which includes data such as electricity price, consumption, production, wind production and cross-border exchange. This data has been extracted and formatted from Nord Pool servers. The models used in this study include linear regression, K-nearest neighbors regression, support vector regression, extreme gradient boosting, and multivariate long short-term memory models. Each model is fitted with a distinct set of hyperparameter combinations, and the resulting performance is evaluated. Once all models have been trained and evaluated according to the specified metrics including mean squared error (MSE), mean absolute error (MAE), root mean squared error (RMSE), and R^2, they are compared to one another in order to draw conclusions.

Glossary

AI – Artificial Intelligence

SDAC – Single Day-Ahead Coupling

ML – Machine Learning

MSE – Mean Squared Error

MAE – Mean Absolute Error

RMSE – Root Mean Squared Error

KNN – K-Nearest Neighbors

SVR – Support Vector Regression

NLP – Natural Language Processing

FTP – File Transfer Protocol

GBM – Gradient Boosting Machine

XGBoosting – Extreme Gradient Boosting

RNN – Recurrent Neural Network

LSTM – Long Short-Term Memory

Adam – Adaptive Moment Estimation

Introduction

This Bachelor's project explores Nord Pool electricity price forecasting through the application of AI algorithms. Electricity price forecasting is a fundamental aspect of modern technology, which has the ability to accurately predict electricity prices based on a multitude of influencing factors. These factors encompass historical data, electricity consumption, production, fuel prices, weather conditions, geopolitical events, and government policies. This ever-evolving innovation is of utmost importance to a diverse range of stakeholders, ranging from consumers and producers to policymakers. Many of the businesses utilize electricity price forecasts in order to enhance the organization and development of their operations. This enables them to gain insights into their electricity consumption patterns.

Nord Pool, one of the largest power market operators in Europe, possesses extensive historical data on electricity prices, consumption, production, and various influencing factors. In addition to their rich historical data, Nord Pool implements their proprietary forecasting methods, providing accurate predictions for electricity prices in the coming days. Traditional forecasting often struggles to capture the complexity of these interactions, encouraging the exploration of cutting-edge technologies such as artificial intelligence (AI) for more dynamic and precise predictions.

The goal of this study is to investigate the optimal AI algorithms for forecasting electricity prices in the Lithuanian Nord Pool exchange market. Additionally, the study aims to enhance the accuracy of price prediction by incorporating various factors that influence price fluctuations.

The task of this study includes:

1. Collect relevant data from the Nord Pool database that will affect electricity prices and reformat it to a suitable data format.
2. Feed the collected data from the Nord Pool database to the AI algorithms.
3. Use different functions and tools to tune the AI algorithms for optimal results.
4. Compare the results of each AI algorithm and evaluate them.

5. Present the process, methodologies, results and conclusions obtained in the work performed, including a list of used literature as the final Bachelor’s work.

1. Analysis

1.1. Noord Pool

Nord Pool has significant influence in the Nordic and Baltic regions. Established in 1993, Nord Pool was the world's first multinational electricity exchange, and it has since grown to become a cornerstone of the European electricity market. In 2013, Lithuania and Latvia joined Nord Pool, thereby further expanding the market's reach and strengthening the integration of the Baltic states into the European electricity market [Nor24b]. Nord Pool operates as a marketplace where electricity producers and consumers trade electricity in the Nordic and Baltic regions. Nord Pool facilitates electricity trading through various marketplaces, including the day-ahead (Elspot) and intraday markets (Elbas) [Nor24a]. The day-ahead market represents a crucial and most essential component of Nord Pool's operations. The market allows market customers to sell or buy electricity for the next 24 hours in a closed auction. The market is of significant importance for balancing supply and demand across the interconnected Nordic and Baltic countries. It also serves to promote cross-border trading and enhance market efficiency. The European market giant possesses an extensive infrastructure and comprehensive historical data on electricity prices, consumption, production patterns and market dynamics. This data is of great value to companies and policymakers. The interconnected nature of the countries allows for cross-border trading, enhancing the efficiency of the electricity market. This wealth of information serves as a valuable resource for companies and policymakers [Nor23a][Nor23b].

The data supports advanced forecasting models, including those using artificial intelligence, to predict future price movements and inform strategic decisions. The transparent publication of market data further empowers participants to make well-informed trading decisions, contributing to a more efficient and resilient electricity market. The implementation of enhanced trading platforms and real-time data analytics have further enhanced Nord Pool's capacity to oversee and optimize electricity trading within a European market that is increasingly integrated and dynamic.

1.1.1. Elspot Market

In the Elspot market, electricity producers and consumers submit their offers and bids on a daily basis. The daily process commences at 10:00 CET, at which time capacity and grid information are released. Participants have until 12:00 CET to finalize their bids for the following day's auction. The matching of orders is carried out using the Euphemia algorithm as part of the Single Day-Ahead Coupling (SDAC) initiative, which optimizes the allocation of electricity across regions based on price and capacity constraints. Hourly clearing prices are typically announced by 12:45 CET, after which individual results are communicated to buyers and sellers. Subsequently, Nord Pool nominates the trades for the imbalance settlement process in each country, thereby ensuring the physical delivery or consumption of the traded energy [Nor23b].

The SDAC initiative is designed to facilitate the smooth and fair trading of electricity across Europe by allowing market participants to submit bids and offers across interconnected regions in a coordinated manner. This integration results in efficient cross-border trading and improves market liquidity while minimizing price differences between neighboring countries. The implementation of this initiative has been made possible through the collaborative efforts of Nord Pool and its partners. They have developed a system that promotes fair competition and ensures market efficiency. The SDAC initiative uses an advanced algorithm, the Euphemia algorithm, to optimize the allocation of electricity across participating regions based on price and capacity constraints. This algorithm ensures efficient market clearing while respecting grid limitations and transmission capacities. As a result, trading costs for participants have been reduced, price transparency has been enhanced and market liquidity has been improved [Nem24].

1.2. Machine Learning

Machine learning (ML) is the development of algorithms that enable computer systems to learn patterns and relationships from historical data, making predictions or decisions without the need for explicit programming or human intervention. It is a powerful technology that is used today, offering a shift from traditional methods. In the context of the Nord Pool market, the application of machine learning algorithms has the potential to significantly enhance the accuracy and adaptability of forecasting models. A number of factors contribute to fluctuations in electricity prices, including market trends, weather conditions and geopolitical events. Machine learning offers a powerful tool for analyzing vast datasets and identifying interconnecting relationships that might surpass traditional methods [Ibm23a].

There are numerous methodologies for forecasting electricity prices that use diverse calculation techniques, each designed to predict and analyze market trends in a distinct manner. These methodologies cover a wide range of approaches, ranging from traditional statistical models such as linear regression to advanced machine learning (ML) algorithms such as neural networks, random forest and support vector machines (SVM). Each algorithm possesses its own unique strengths, allowing for extensive exploration of the unpredictable nature of electricity markets. Upon being trained on Nord Pool's historical data, these algorithms are capable of uncovering intricate patterns, adapting to fluctuating market conditions, and providing invaluable insights for the accurate forecasting of electricity prices.

In machine learning, there are two primary types of models exist: classification and regression. These models serve different purposes based on the nature of the prediction problem and the task at hand. Classification models are designed to assign input data to a category or a class, making them suitable for tasks where the output is categorical. Examples include image recognition where objects are identified in a photograph. In contrast, regression models are used for predicting continuous numerical values. This type of model is utilized when the output variable represents a value, such as in the prediction of electricity prices or the estimation of temperatures. The evaluation of regression models includes measures such as mean squared error (MSE), mean absolute error (MAE), and others, which calculate the accuracy of predictions against actual values. In summary, this project will follow regression models [Vih23][Ant23].

Regression models are a type of supervised learning algorithm. In supervised learning, the algorithm is trained with both input features (also known as predictors) and output target labels.

The algorithm's task is to learn the relationship between the inputs and the outputs so that it can accurately predict the target labels for new, unseen data [Ger17, Chapter 1, p. 8-9].

1.2.1. Linear Regression

Linear regression model was developed in the field of statistics, which is used to understand the relationship between input and output of numerical values. This statistical algorithm has been borrowed by machine learning, making it both a statistical and machine learning algorithm. This machine learning model is one of the easiest models to apply and use. The algorithm is simple and used for predicting continuous outcomes based on one or more input features represented by a linear equation, making it easy to understand and visualize. The linear regression model is widely utilized in a multitude of fields, including biology, social science, epidemiology, and finance studies. The objective of this model is to identify the optimal linear fit that minimizes the discrepancy between the predicted and actual values [Ger17, Chapter 4, p. 106-110]. The fundamental equation of linear regression is represented as follows:

$$y = b_0 + b_1 x$$

In this context, y represents the dependent variable (output variable), while x denotes the independent variable (input variable). The term b_0 refers to the interception of the y-axis, and b_1 denotes the slope of the line, which represents the change in y with respect to x. The term b, in this context, is used to refer to the coefficients, which represent the weight or contribution of the corresponding independent variable to the prediction of the dependent variable. In the context of multiple linear regression, when there are multiple input values, the equation becomes:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n$$

These coefficients are estimated during the training process, during which the algorithm adjusts their values in order to minimize the difference between the predicted and actual values. For the purpose of learning a linear regression model, it is of vital importance to estimate the values of the coefficient. There are multiple ways of doing this, such as for simple linear regression, it is sufficient to calculate statistical properties from the data, such as means, standard deviations, correlations and covariance. For multiple linear regression, coefficients can be calculated using ordinary least squares, gradient descent or regularization [Bro23a].

1.2.2. K-Nearest Neighbors (KNN) Regression

K-Nearest Neighbors (KNN) is a simple and widely used model in machine learning for classification and regression tasks. While it is typically used as a classification model, it can also be applied to regression tasks. The distance to the nearest point (neighbor) is used to make classifications or predictions about the grouping of the individual data points. KNN classification is used to predict the label or categorical class of a new instance, providing an indication of the category to which the input value belongs. The KNN regression model uses a similar concept to the classification model, however, its objective is to predict a continuous numeric value [TTM23].

The KNN algorithm operates by initially gathering data from the dataset, which includes input features and target values. For KNN regression, the target values are continuous values and represent the output to be predicted. When making predictions, KNN uses three hyperparameters: the parameter K, the distance metric and the weights. The parameter K represents the number of nearest neighbors used for prediction. The distance metric is used to measure the distance between the data points. Different distance metrics can be used to generate different outputs [Bro23b][Ibm23b]. The parameter weights are used to assign varying degrees of significance to the neighbors in the process of making predictions. There are two ways to assign weights to the neighbors. The first is uniform weights, wherein all neighbors contribute equally to the prediction. The second is Distance weights, wherein closer neighbors are given more weight in the prediction than those further away. Figure 1 provides an illustration of the KNN example.

While these hyperparameters can be manually controlled, there are hyperparameter tuning methods that select the most suitable parameters for the data. A more detailed discussion of hyperparameter tuning will be presented in this chapter. In the KNN regression method, the predicted value for the new data point is the average of the target values of the K nearest neighbors.

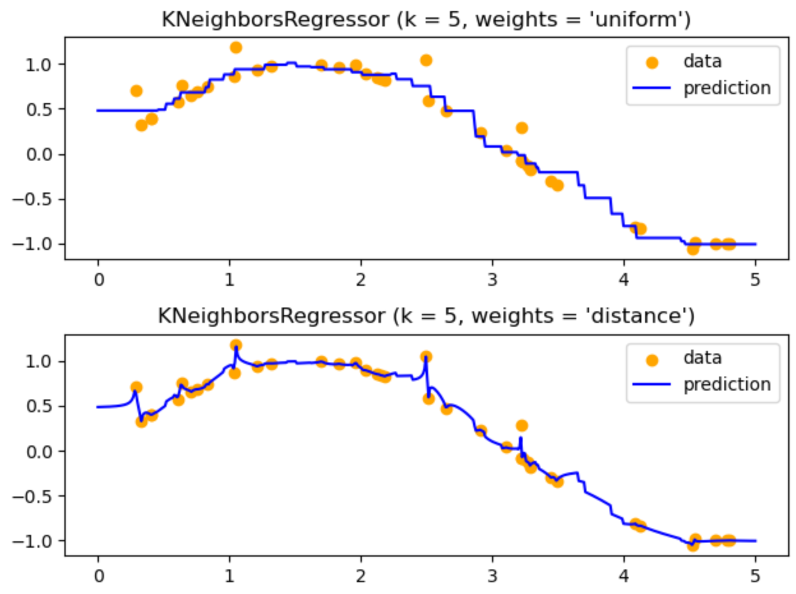

Figure 1: K-Nearest Neighbor Regression graphical representation [GP24].

1.2.3. Support Vector Regression (SVR)

Support Vector Regression (SVR) is a machine learning algorithm that extends from the Support Vector Machine (SVM) and is designed to predict continuous numeric values [Ger17, Chapter 5, p. 154-156]. As a supervised learning approach, it is a suitable model in the context of Noord Pool's electricity price forecast. SVR uses a symmetric loss function during training and penalizes both high and low estimation errors equally. Vapnik's ε-insensitive approach to SVR construction entails the formation of a flexible tube around the estimated function, with a minimized radius. This is referred to as an ε-tube, within which errors smaller than a certain threshold ε are disregarded, both above and below the estimated function. Points outside this tube are penalized, while those within it face no penalty. Figure 2 provides an illustration of the SVR functionality. This tube facilitates the reshaping of the optimization problem, with the objective of identifying the optimal tube version that approximates the continuous-valued function while considering model complexity and prediction error. The goal is to identify the flattest tube that encompasses the majority of training instances. Consequently, a multiobjective function is constructed, combining the loss function with the geometric attributes of the tube. The hyperplane is expressed in terms of support vectors, which represent training samples located outside the tube boundary. These support vectors play a crucial role in determining the tube's shape in SVR [AK15]. The basic equation of SVR is represented as follows:

$$min\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

In this context, $\|w\|^2$ is the squared loss between the predicted and actual values, C is the regularization parameter that controls the trade-off between minimizing the error and maximizing the margin. ξ_i and ξ_i^* are slack variables that allow some points to fall outside the ε-insensitive tube.

This model has numerous hyperparameters that can be configured, resulting in a broad spectrum of potential outcomes. These hyperparameters consist of kernel functions, regularization parameters (c) and epsilon (ε). Hyperparameter tuning is of paramount importance for this model to achieve the most accurate results. Kernel functions are used for transforming the feature space, creating the modeling of non-linear relationships between input features and target variables. The regularization parameter serves to regulate the balance between maximizing the margin and minimizing the prediction error. The epsilon parameter defines the width of the epsilon tube, which represents the range around the regression line where deviations are permitted, subject to a penalty [Ver23][WH23].

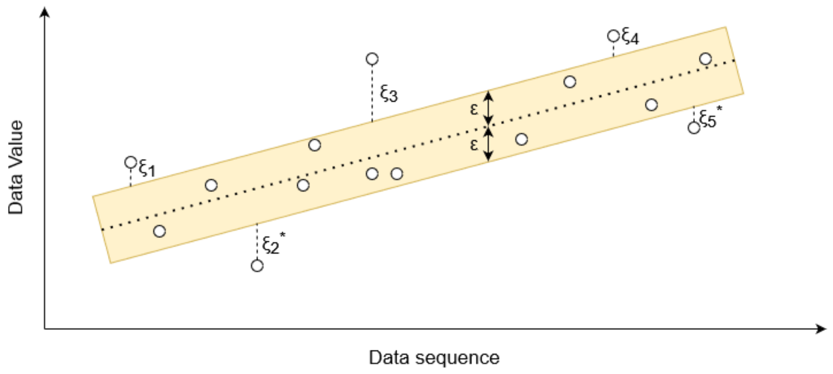

Figure 2: Support Vector Regression graphical representation.

1.2.4. Gradient Boosting Machine (GBM)

Gradient Boosting Machines are a class of machine learning algorithms that are powerful tools for solving regression and classification problems. They are a combination of learning methods, meaning that they combine the predictions of several individual models to produce the final prediction [Bro20c][NK13]. Gradient Boosting features include:

- **Boosting** is a method of transforming weak learners into strong learners, by building a sequence of weak learners, like a tree, that corrects the errors of its predecessor. These methods build trees sequentially and each subsequent tree in the sequence focuses on the samples that were misclassified by the previous trees.
- **Gradient descent** refers to the optimization algorithm used to minimize the loss function. In each iteration, the algorithm fits a new weak learner to the residual errors of the previous models. It fits the models by calculating the gradient of the loss function in the negative direction to minimize the loss. The loss function is a quantitative measure of the discrepancy between the model's predictions and the actual target values.
- **Combining weak learners** results in a reduced error. This happens after training all the weak learners and then it is combined to make the final output. A weak learner is a simple model that performs slightly better than random chance on a given problem. While it may not be highly accurate on its own, it can contribute to the overall performance when combined with other weak learners in techniques such as boosting.

1.2.4.1. Extreme Gradient Boosting (XGBoost)

This specific model will be used in this project. Extreme Gradient Boosting is an optimized and highly efficient implementation of gradient boosting machines. It is the most common model from the Gradient Boosting family and it is widely used for structured and tabular data.

XGBoost is highly optimized for performance, making it much faster than traditional GBM implementations. It is capable of handling large datasets with high dimensionality and is robust to overfitting, coupled with its ability to maintain high predictive accuracy. This model was created for the sole purpose of efficient computation. There are three important hyperparameters for XGBoost: the number of trees, the tree depth, and the learning rate. The first is the number of trees, which are added to the model sequentially, resulting in improved predictions made by prior trees. The second is the tree depth, which captures more patterns the deeper the tree goes. The third is the learning rate, which determines the robustness of the model to overfitting. Smaller values make the model more robust to overfitting but may require more trees to model the data effectively [Bro21a].

Other features that contribute to XGBoost's efficiency include built-in regularization techniques such as ridge and lasso regularization. Ridge regularization tends to shrink the coefficients towards zero, but it rarely sets them exactly to zero. In contrast, lasso regularization encourages sparsity by setting some coefficients exactly to zero. The ridge is a useful technique when all the features are relevant, whereas the lasso is useful for feature selection and when dealing with datasets with many irrelevant features [Bro16].

1.3. Artificial Neural Networks

Artificial Neural networks (ANN) are computational models inspired by the structure and function of the human brain. This machine learning process is called deep learning, which consists of interconnected nodes, called neurons, organized in layers, similar to the human brain. This system is designed to learn from mistakes and find ways to improve itself.

This technology is versatile, powerful and scalable and is used to solve complex problems in areas such as image and speech recognition, natural language processing, autonomous vehicles and more. It is a powerful tool for learning complex patterns and applying them to solve real-world problems [Ibm24a].

As a neural network is trained, it learns to adjust these weights based on the input data and the desired output. This process involves feeding input data through the network, calculating the output, comparing it to the desired output, and then using an optimization algorithm. Once trained, a neural network can be used to make predictions on new, unseen data [Ger17, Chapter 10].

This project uses a type of artificial neural network called a recurrent neural network (RNN), which is designed to process sequential data or time series, where the order of the information is significant. They are used in natural language processing (NLP), speech recognition, and in this project's case, time series analysis. RNNs are distinct from other neural networks in that they can retain memory or context from previous inputs through loops within the network architecture. This memory mechanism allows RNNs to maintain information persistence over time, rendering them well-suited for tasks involving sequential data [Ger17, Chapter 14].

The basic operation of an RNN involves processing input sequences one element at a time, updating its internal state (also known as the "hidden state") at each time step based on the current input and the previous hidden state. This updated hidden state then influences the output produced by the network at that time step, as well as serving as input for the next time step [Ibm24b].

1.3.1. Long short-term memory (LSTM)

Long short-term memory shortly known as LTSM is a type of RNN architecture designed to address the vanishing gradient problem, which is common in traditional RNNs. Its goal is to provide the RNN with a short-term memory that can last for thousands of timesteps, hence the name “long short-term memory”. This algorithm is applicable to classification, processing and prediction of data based on time series and is commonly used in handwriting, speech recognition, machine translation, etc. LSTM is specifically designed to handle sequential data and can capture long-term dependencies, it is expected to perform well on this type of data.

LSTM is constructed from fundamental components, including cells and gates. Figure 3 provides an illustration of the LSTM cell. The following section provides a step-by-step guide on how they work.

The cell state is the memory of the LSTM. It runs straight through the entire chain of LSTM cells, with only minor linear interactions. Information can be added to or removed from the cell state by structures called gates, allowing the LSTM to selectively remember or forget information over time.

The first step is the **forget gate**. This is a sigmoid layer that takes input from the previous hidden state and the current input and outputs a number between 0 and 1 for each number in the cell state. A value of 1 signifies the retention of the information, whereas a value of 0 indicates its discarding. This gate determines which information from the cell state should be discarded or retained.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

f_t is the forget gate output at the time step t,
σ is the sigmoid activation function,
W_f is the weight matrix for the forget gate,
h_{t-1} is the previous hidden state,
x_t is the current input,
b_f is the bias vector for the forget gate.

The second step is the **input gate**. This consists of two distinct parts: a sigmoid layer and a tanh layer. The sigmoid layer determines which values to update, while the tanh layer generates a vector of new candidate values to add to the cell state. These two are then combined to create an update to the state.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

i_t is the input gate output at the time step t,
W_i is the weight matrix for the input gate,
b_i is the bias vector for the input gate.

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

$\tilde{C}_t$ is the candidate cell state at the time step t,
W_C is the weight matrix for the candidate cell state,
b_C is the bias vector for the candidate cell state.

The third step is to **update** the old cell state C_{t-1}, into the new cell state C_t.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

C_t is the updated cell state at the time step t,

C_{t-1} is the previous cell state.

The fourth and final step is the **output gate**. This determines the information to be output based on the updated cell state. It takes input from the current input and the previous hidden state, and outputs the relevant parts of the cell state. Then, the cell state was subjected to tanh and then multiplied by the output of the sigmoid gate. Tanh confines the values to the range of −1 to 1.

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

o_t is the output gate output at the time step t,

W_o is the weight matrix for the output gate,

b_o is the bias vector for the output gate.

$$h_t = o_t * tanh(C_t)$$

h_t is the current hidden state.

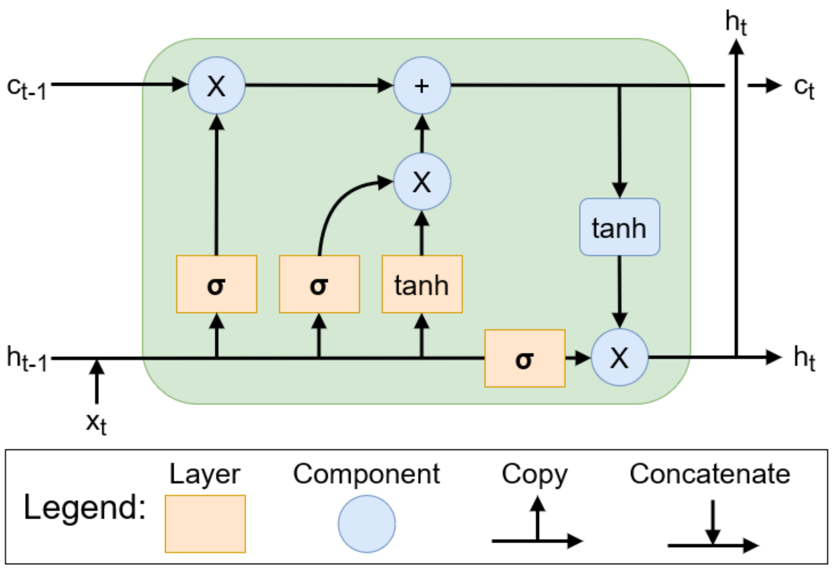

Figure 3: Long Short-Term Memory cell example.

This sequence of computations enables the LSTM cell to selectively retain or discard information from the input data and previous hidden state, thereby producing the appropriate output for the current time step [Ger17, Chapter 14, p. 402-406][Ola24].

The LSTM is trained using the Adaptive Moment Estimation (Adam) optimizer. This is a stochastic gradient descent optimizer that uses adaptive learning rates. It is a type of gradient descent optimizer that uses a moving average of the gradient to determine the learning rate [Bro21b].

1.3.1.1. Multivariate LSTM Forecast Model

This project will use the LSTM model for multivariate time series forecasting. This is a highly effective technique that allows the prediction of multiple time series variables simultaneously, taking into account their interdependencies. In this context, each input time step may consist of multiple features or variables, and the LSTM model will learn to capture the relationships between these variables over time to make accurate predictions. Later in the project, the data will comprise multiple variables that will be used in the prediction of electricity prices. Consequently, multivariate time series forecasting is an appropriate methodology for the implementation and application of LSTM. The anticipated outcome of multivariate LSTM is comparable to that of the original LSTM, as it is well-suited for detecting sequential data. In essence, the model should yield satisfactory results by incorporating features for prediction. [Bro20a][Bro20b].

The architecture of the model remains largely consistent with that of univariate time series forecasting. However, a few key differences will be apparent in the Multivariate LSTM forecast model:

The input data preparation process involves organizing the data into a 3D array where the dimensions represent the number of samples, time steps, and features. This format allows the LSTM model to process multiple variables at each time step.

As this project will use LSTM models for multivariate time series forecasting, it is necessary to convert the time series data into a supervised learning problem. This allows the model to learn the mapping between input variables (features) and output variables (target). By framing the problem as supervised learning, you have the flexibility to incorporate additional features or variables into your model that may help improve its predictive performance. This process involves transforming the sequential observations into a format where each input sample is associated with a corresponding target variable. This typically entails creating a dataset where each row represents a sequence of lagged observations of the input variables, and the target variable is the observation at the next time step. From this, the LSTM is able to learn to predict future values based on historical patterns and relationships within the time series data.

Modification of the architecture of the Multivariate LSTM model need to be adjusted to accommodate the multivariate nature of the data. This may include modifying the number of input nodes.

The Output of the Multivariate LSTM will also be in a multivariate format, where each output time step corresponds to predictions for multiple variables. As with the input data, the output data should be organized into a 3D array.

The Loss Function will be used for training the multivariate time series forecasting LSTM model. The two available choices are the mean squared error (MSE) and the mean absolute error (MAE), which are calculated across all predicted variables.

The Evaluation metrics used in univariate forecasting may differ from those used in this context. Therefore, it is important to consider metrics that take into account the correlation between predicted and actual values. Such metrics include the Mean Absolute Percentage Error (MAPE) and the Pearson correlation coefficient.

1.4. Machine Learning Modifications

1.4.1. Distance-Based Models

The KNN model uses a distance function to work as it is a distance-based model. Consequently, it is crucial to examine a range of distance metric options when utilizing distance-based models.

1.4.1.1. Euclidean distance

Euclidean distance is the shortest distance between two data points in a straight line. It is a simple and widely used distance metric in machine learning and various other fields, including geometry and physics. Euclidean distance is represented in two-dimensional space as follows:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The distance (d) is calculated using the formula above, which uses two points: $q(x_1, y_1)$ and $p(x_2, y_2)$. The Euclidean distance in n-dimensional space is represented as follows:

$$d = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

The dimension of the feature space, n, is a crucial factor to consider when normalizing feature vectors. A feature value with an exceedingly high value will have a significantly greater impact on the outcome than a low value [Gor21][SGR21].

1.4.1.2. Manhattan distance

The Manhattan distance, also known as the taxicab distance, is a distance between points in a grid-based system. It is calculated as the sum of the absolute difference between the x and y coordinates. The Manhattan distance is represented in two-dimensional space as follows:

$$d = |x_1 - x_2| + |y_1 - y_2|$$

The coordinates (x_1,y_1) and (x_2,y_2) represent points in space. The Manhattan distance is represented in n-dimensional space as follows:

$$d = \sum_{i=1}^{n} |x_{1i} - x_{2i}|$$

The variable n represents the number of dimensions or features present in the given space. The x_{1i} and x_{2i} variables represent the i-th coordinate of the two points [Sah21].

1.4.1.3. Minkowski distance

The Minkowski distance is a combination of Euclidean distance and Manhattan distance. Minkowski distance is used as a measure a similarity between two data points. The Minkowski distance is represented as:

$$d = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

The variable n represents the number of dimensions or features present in the given space. p is a parameter that defines the order of the Minkowski distance. When $p = 1$, it is equivalent to the Manhattan distance, and when $p = 2$, it is equivalent to Euclidean distance. The x_i and y_i variables represent the i-th coordinate of the two points [Tur22].

1.4.2. Splitting data

There are numerous methodologies for splitting a dataset for the purposes of training and testing machine learning algorithms. The simplest and most common choice is splitting the dataset into two parts: a training set and a testing set. Other techniques for splitting datasets include cross-validation and random shuffling, which is then followed by the split of the dataset into training and testing subsets. This project uses a time series dataset, and it is customary to partition the available data into two distinct subsets: a training set and a testing set. This partition is necessary to maintain the chronological order of the data. The training set encompasses historical observations preceding the testing set, ensuring that the model is trained on past data to capture underlying patterns and dynamics. By dividing the data into training and testing sets while preserving chronological order, the integrity of the temporal structure is maintained.

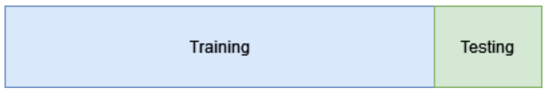

Figure 4: Data split representation into a training set and testing set.

1.4.3. Hyperparameter tuning

Hyperparameter tuning is an important aspect of developing a machine learning model. This method aids in identifying the optimal set of hyperparameters for a given model, thereby achieving the most optimal results. Hyperparameters are not derived from the data and represent a configuration setting for the model. Each configuration can influence the model in a distinct manner and identifying the optimal values often involves a process of trial and error.

1.4.3.1. Grid Search

Grid search is a hyperparameter optimization technique used in machine learning to systematically explore a predefined set of hyperparameters and their respective values for a given model in a grid-like fashion, testing through all possible combinations. For each combination of hyperparameters, the model is trained and evaluated using a specified evaluation metric, such as accuracy, precision, or mean squared error [Ger17, Chapter 2, p. 72-74]. The primary objective of grid search is to identify the optimal combination of hyperparameters that maximizes the performance of the model on unseen data. By systematically exploring different hyperparameter configurations, grid search helps to fine-tune the model and improve its generalization capability [Jos18].

1.4.3.2. Manual Search

This technique involves manually running a model training through every hyperparameter sequence to identify the most optimal variant. This method will be used to identify the optimal hyperparameter sequence for the LSTM model. This will facilitate the systematic exploration of different combinations of hyperparameters and the evaluation of their performance on a validation dataset.

2. Design and Implementation

The objective of this chapter is to examine the strategic choices made in the construction of a robust framework for Nord Pool electricity price forecasting through the integration of machine learning algorithms. This section outlines the methodologies and data considerations to address the electricity market, with a focus on achieving accurate results.

2.1. Methodologies

2.1.1. Python

The prototype will be developed using the Python programming language due to its simplicity, easy-to-learn syntax and extensive libraries. Python provides a versatile environment, offering a wide range of machine learning algorithm libraries that are valuable for developing the prototype.

2.1.2. Libraries

The prototype will be developed using a wide range of Python libraries:

- **Sckit-learn** is a robust machine learning library that provides a comprehensive array of machine learning algorithms and tools for data analysis and modeling. This library will be used to implement and experiment with a diverse range of machine learning algorithms.
- **Matplotlib** is a comprehensive two-dimensional plotting library that creates a visualizations in Python. It is commonly used to plot diagrams or charts, which are essential for visualizing data. This library will be used to visualize the Noord Pool data and assess machine learning algorithm results.
- **NumPy** is a fundamental package for scientific computing. It provides support for handling large, multi-dimensional arrays, matrices and tools that assist in the efficient calculation of data. This library will be used to perform mathematical operations on extracted data.
- **Pandas** is a library for data manipulation and analysis. It provides data structures and functions that are straightforward to use, allowing users to work with structured data,

such as tables and time series. Pandas is built on top of NumPy, offering high-performance, labeled data structures like DataFrame and Series, which allow users to perform operations with ease. This library will be used to extract datasets from our data files and manipulate them.

- **Seaborn** is a statistical data visualization library built upon Matplotlib. It provides tools that create statistical graphs, such as heatmaps, violin plots and others. This library will be used to create a correlation matrix of the data.
- **Keras** is a high-level neural network library written in Python that serves as an interface for artificial intelligence research and experimentation. It provides an easy-to-use and flexible platform for building, training, and deploying deep learning models.

2.2. Data Considerations

The extensive database displayed on the Noord Pool website is not available for download. Only electricity prices are publicly accessible for download. This project requires multiple factors to forecast electricity prices, and thus additional data is needed. There is an option to subscribe to their services for viewing and accessing the complete database, but it comes at a significant cost. Consequently, the author of this Bachelor's project has taken the initiative of contacting the Noord Pool team with a request for access to their database as a student at Vilnius University. Fortunately, Nord Pool has agreed to provide access to the database through the Nord Pool file transfer protocol (FTP) server.

The majority of data stored on an FTP server is in the SDV file format. SDV stands for semicolon-divided values, indicating that each cell is divided by semicolons. The majority of SDV files contain data from a single week. To obtain data spanning a year, it is necessary to combine these files. For this project, SDV files will be converted into an Excel file for visualization purposes and as input for machine learning algorithms. The newly reformatted Excel file will have hours as rows and features as columns. The data that will be used in this project will consist of:

- Price of electricity in Lithuania (EUR/MWh)

- Electricity production in Lithuania (MWh/h)
- Wind electricity production in Lithuania (MWh/h)
- Electricity consumption in Lithuania (MWh/h)
- Cross-border exchange between Lithuania and bordering countries (MWh/h).

Regarding cross-border exchanges between Lithuania and bordering countries, the values of each exchange in the neighboring countries will be summed and added to a single column. This approach is intended to reduce the number of unnecessary features and to facilitate the training of more efficient and faster models.

The time period of this data encompasses the period from January 1, 2021, to December 31, 2023. An illustrative data format is presented in the table below.

Date	Hour	Price	Consumption	Production	Wind production	Cross-border exchange total
01.01.2021	00 - 01	12.34	1234	1234	1234	1234
01.01.2021	01 - 02	12.34	1234	1234	1234	1234
…						
31.12.2023	23 - 00	12.34	1234	1234	1234	1234

Table 1: Nord Pool data representation in Lithuania example.

The following figures present graphical representations of the data for the period from January 1, 2021, to December 31, 2023.

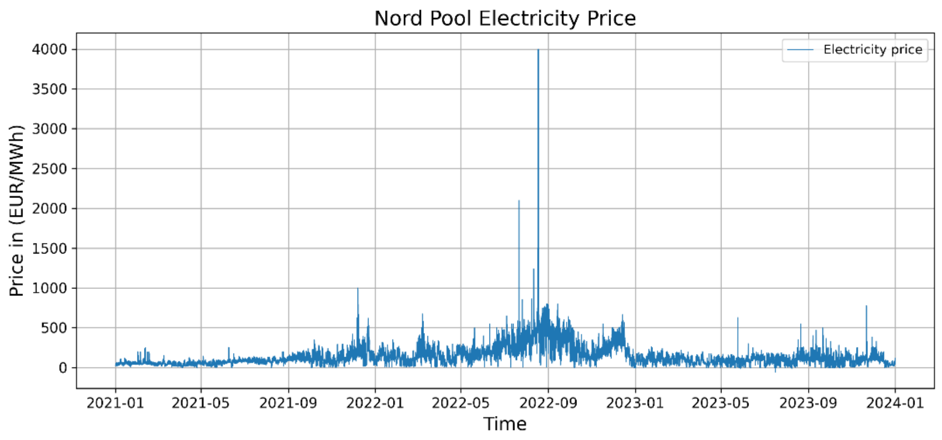

Figure 5: Nord Pool Electricity Price Data Representation in Lithuania.

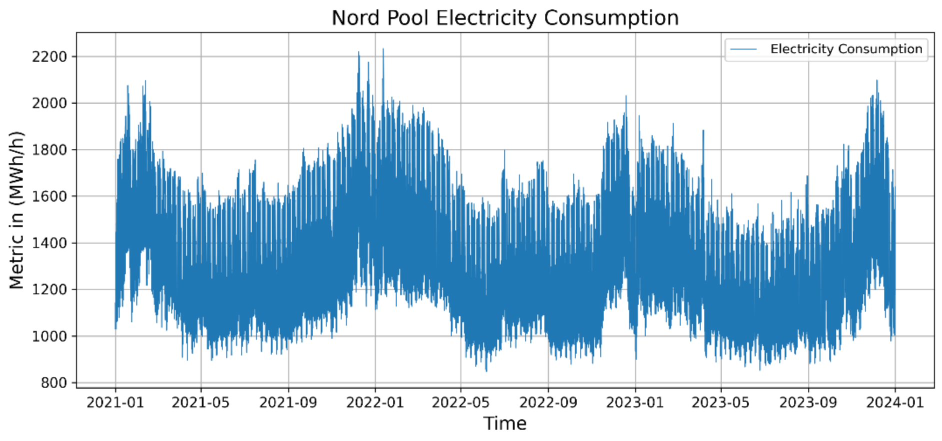

Figure 6: Nord Pool Electricity Consumption Data Representation in Lithuania.

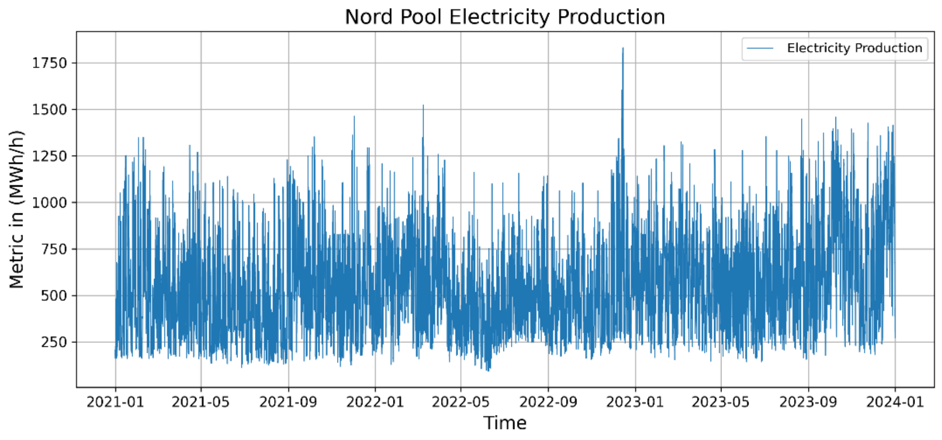

Figure 7: Nord Pool Electricity Production Data Representation in Lithuania.

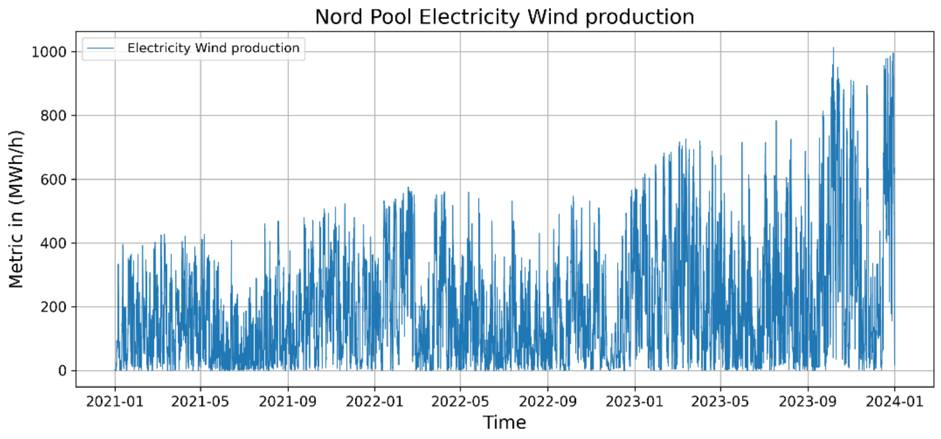

Figure 8: Nord Pool Electricity Wind Production Data Representation in Lithuania.

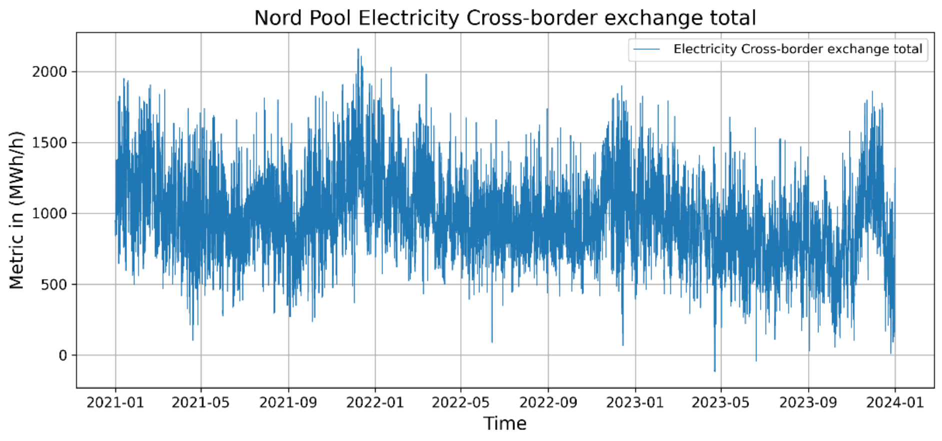

Figure 9: Nord Pool Electricity Cross-border Exchange with Neighboring Countries Data Representation in Lithuania.

2.3. Model Training

To proceed with model training, first, it is essential to import the dataset and establish the target values and feature values. The dataset is extracted from the Excel file. In this context, the electricity price will serve as the target value, while other features such as production, consumption, wind production and cross-border exchanges will be utilized as the feature values.

Next, the dataset must be split into training and testing sets using the train_test_split function from the Sckit-learn library. The project will follow a split of 80% for the training set and 20% for the testing set. It is important to note that the dataset is a time series based dataset and therefore, the dataset should not be shuffled.

Subsequently, the model is chosen from the Sckit-learn library and the data is fed into the machine learning algorithm. The model is trained and outputs metrics and the trained model itself.

Furthermore, as a pivotal aspect of this project, the optimization of algorithm performance involves selecting the best hyperparameter combinations. To achieve this, a grid search will be used to identify the optimal hyperparameter combinations for each algorithm. The grid search function is imported from the Sckit-learn library and will be used to determine the most optimal hyperparameter combination. Evaluation metrics will be used to determine the optimal hyperparameter combination. These metrics will be presented later in this project. In order to perform the grid search, the mean squared error metric will be employed.

2.3.1. Model training for multivariate LSTM

Training a Multivariate LSTM requires different steps than traditional methods. To proceed with LSTM training, import the dataset from the Excel file and set the target value and feature values. Set the electricity price as the target value and other features as the feature values.

Next, it is important to frame the dataset as a supervised learning problem and to normalize the input variables. This is achieved by creating a function that would convert time series into a supervised learning problem. This process involves converting sequenced data into input and output sequence data. The process involves the construction of two distinct sets of data: lag observations and forecast observations. In this project, the lag coefficient will be set to one hour. This means that feature values and target values will be used to predict the target value in one hour's time. Additionally, it is crucial to normalize the dataset because LSTM is highly sensitive to large value changes.

Then, the dataset must be split into a training set and a testing set using the train_test_split function from the Sckit-learn library. The split ratio is 80% training set and 20% testing set. It should be noted that this data is time series-based and no shuffle is required.

Subsequently, the model is selected from the Keras library and the data is fed into the LSTM model. The model is trained from the given dataset with hyperparameters such as neuron, epoch and batch counts, as well as loss metrics and optimizers. The trained model and its accuracy are then outputted. The LSTM model is trained with the Adam optimizer. The evaluation metrics that will be presented later in the project will allow for the comparison and assessment of the same LSTM model with different hyperparameter combinations, as well as comparisons with other models.

Furthermore, as a crucial component of this project, the LSTM model must undergo distinct training with the same dataset, but with different hyperparameter configurations. For the LSTM model, hyperparameter tuning via grid search is not a viable approach, as it is essential to conduct multiple iterations of the same LSTM model training with the same hyperparameters to obtain average evaluation metrics and then compare the LSTM model with different hyperparameter configurations. Consequently, a script will be created to evaluate each hyperparameter sequence in a grid search-like manner. This script will run 10 times for each of the hyperparameters in range and will output the average mean, standard deviation, minimum, and maximum values for each evaluation metric, as well as a graphical representation. The outcome of this process will determine which of the hyperparameter sequences yields the most accurate metric evaluation. Subsequently, the optimal and most precise model for forecasting Nord Pool electricity prices will be identified. This will be accomplished by combining the hyperparameters that have demonstrated the greatest performance and training them 10 times to obtain the average evaluation metrics and a graphical representation.

3. Requirements

The objective of this chapter is to define the criteria that will be utilized to assess the performance of the machine learning model [Ger17, Chapter 2, p. 37-40].

3.1. Evaluation Metrics

In order to assess the efficacy of each model, a series of predefined metrics will be established and subsequently compared [Baj23][Dev23].

The **Mean Absolute Error (MAE)** is an average of the absolute difference between predicted and actual values. It provides an idea of the kind of effects errors display without considering direction. The formula is as follows:

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

The notation n is the number of observations in a given dataset, y_i represents the actual value and $\hat{y}_i$ represents the predicted value.

The **Mean Square Error (MSE)** is an average of the squared difference between predicted and actual values. It provides a penalty on larger errors that is harsher than that of MAE. The formula is as follows:

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

The notation n represents the number of observations in a given dataset, y_i represents the actual value and $\hat{y}_i$ represents the predicted value.

The **Root Mean Square Error (RMSE)** is the square root of the MSE. It corresponds to the square root of the average of the squared difference between the target value and the value predicted. It provides an interpretable scale in the same units as the target variable. The formula is as follows:

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

The notation n represents the number of observations in a given dataset, y_i represents the actual value and $\hat{y}_i$ represents the predicted value.

The **R-square ($\boldsymbol{R^2}$)** represents the proportion of the variance in the dependent variable that can be predicted from the independent variables. The value ranges between 0 and 1, with 1 indicating a perfect fit. It can also take on negative values, which indicates that the model fit is worse than that of a straight line. The formula is as follows:

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

The notation n represents the number of observations in a given dataset, y_i represents the actual value, $\hat{y}_i$ represents the predicted value and $\bar{y}$ represents the mean of the observed values.

3.2. Finding the Best Hyperparameters

In order to identify the optimal hyperparameters for a machine learning model, a grid search will be utilized. For the Multivariate LSTM model, a manual search will be conducted. Each model will possess a distinct set of hyperparameters.

3.2.1. Linear Regression Hyperparameters

Linear regression does not possess any hyperparameters, therefore grid search will not be applied to the linear regression model.

3.2.2. K-Nearest Neighbors (KNN) Regression Hyperparameters

The process of KNN regression hyperparameter optimization will consist of the following steps:

- K: [10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]
- Distance metric: [Euclidean, Minkowski, Manhattan]
- Weights: [uniform, distance]

3.2.3. Support Vector Regression (SVR) Hyperparameters

The process of SVR regression hyperparameter optimization will consist of the following steps:

- Kernel: [linear, rbf, poly]
- C: [0.1, 1, 10]
- Epsilon: [0.01, 0.1, 1]
- Gamma: [0.0001, 0.001, 0.01]

3.2.4. XGBoosting Hyperparameters

The process of XGBoost hyperparameter optimization will consist of the following steps:

- Max depth: [3, 5, 7, 9]
- Learning rate: [0.01, 0.05, 0.1]
- N estimators: [100, 300, 500, 1000]

3.2.5. Multivariate LSTM Hyperparameters

The process of Multivariate LSTM hyperparameter optimization will consist of the following steps:

- Epochs – [20, 50, 100]
- Batch size – [32, 64, 128]
- Neurons – [50, 100, 150, 200]

4. Results

4.1. Correlation Matrix

The correlation matrix was created using the Seaborn Python library and is presented below. Upon examination of the correlation matrix, it becomes apparent that consumption and production features are highly dependent on each other. Additionally, wind production shows a significant correlation with production. There are negative correlations visible, such as production having a significant negative correlation with total cross-border exchange. Upon further examination, it becomes evident that price has no significant correlation with any of the features. The highest positive correlation is observed with consumption, with a value of 0.27, while the highest negative correlation is observed with wind production, with a value of -0.24. It is important to note that correlation values range from -1 to 1. A value of -1 indicates a perfect negative correlation, while a value of 1 indicates a perfect positive correlation, and a value of 0 indicates no correlation.

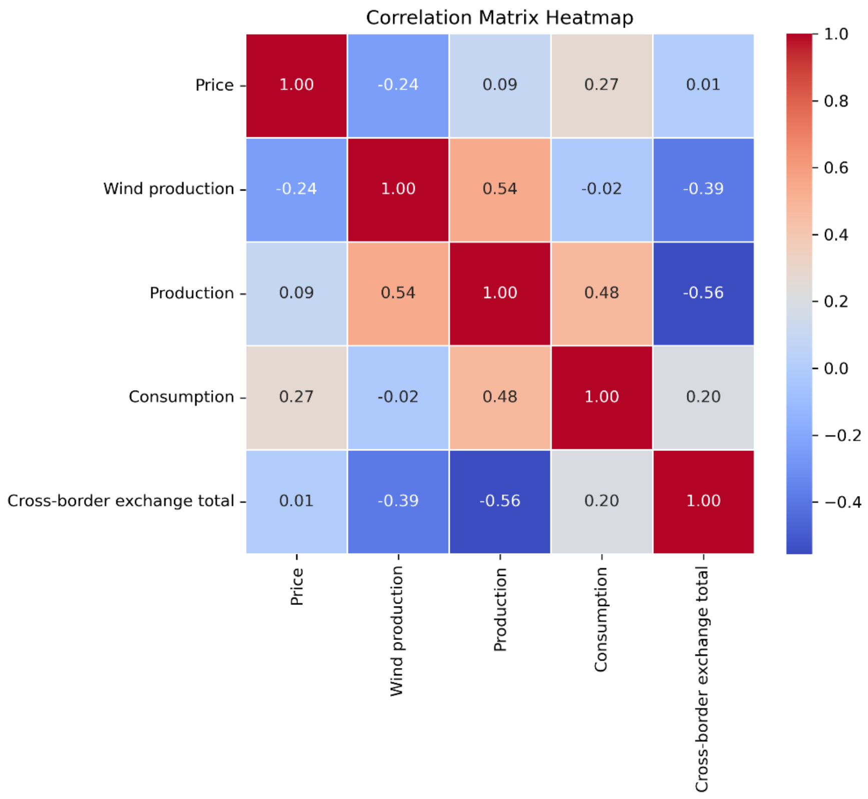

Figure 10: Correlation matrix representation of the Nord Pool dataset.

4.2. Model Results

4.2.1. Linear Regression Model Results

This section presents a comprehensive metric evaluation of the Linear Regression model, accompanied by a graph illustrating the relationship between actual electricity prices and predicted electricity prices.

MSE	MAE	RMSE	R^2
4364.0732	51.9762	66.0611	-0.2625

Table 2: Metric evaluation of the Linear Regression model.

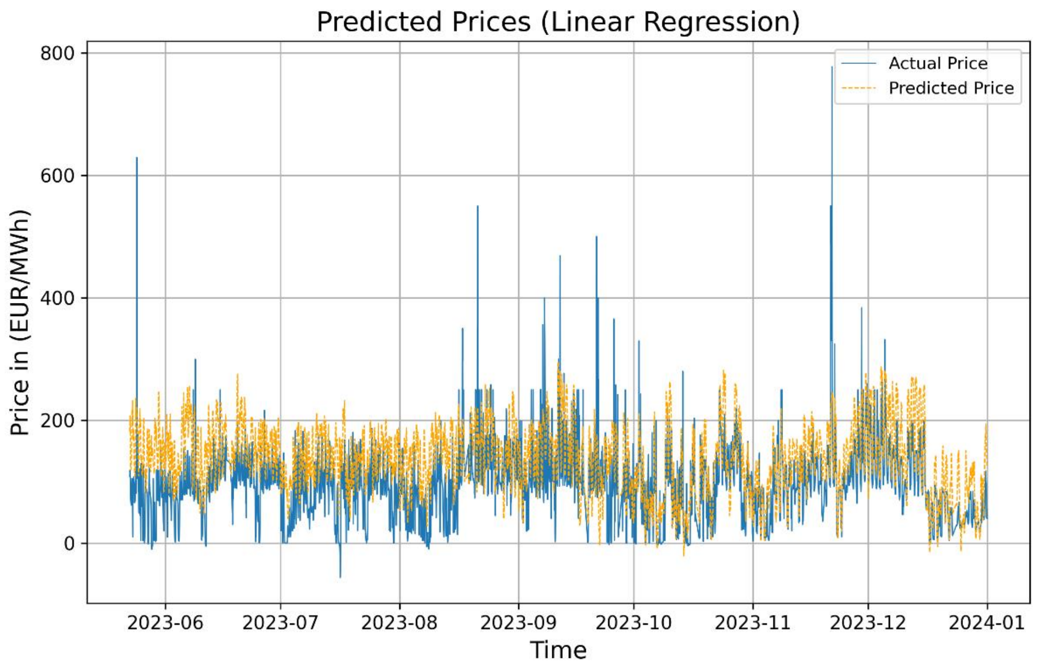

Figure 11: A graphical representation of the Linear Regression models predicted electricity price versus actual electricity price.

4.2.2. K-Nearest Neighbors (KNN) Regression Model Results

This section presents the optimal hyperparameter configuration for the K-Nearest Neighbors Regression model.

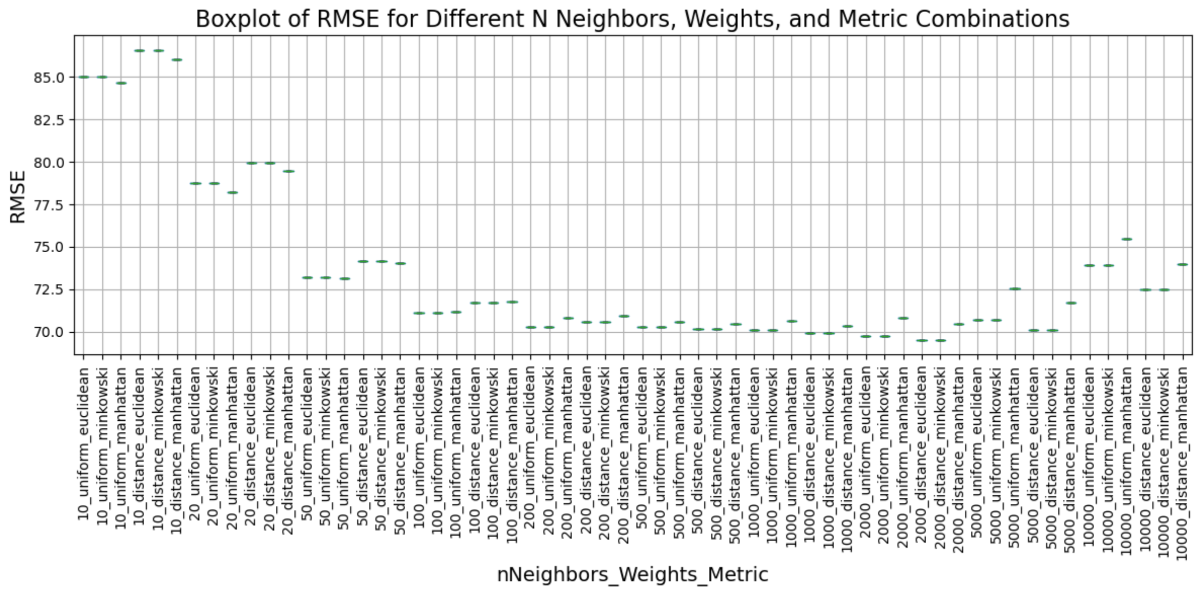

Figure 12: A graphical representation of the KNN Regression hyperparameter combinations.

The optimal hyperparameters for KNN Regression, as determined through grid search, are as follows:

- K: 2000
- Distance metric: Euclidean distance
- Weights: distance

This section presents a comprehensive metric evaluation of the optimal K-Nearest Neighbors Regression model, accompanied by a graph illustrating the relationship between actual electricity prices and predicted electricity prices.

MSE	MAE	RMSE	R^2
4835.1372	57.1536	69.5252	-0.3988

Table 3: Metric evaluation of the KNN Regression model.

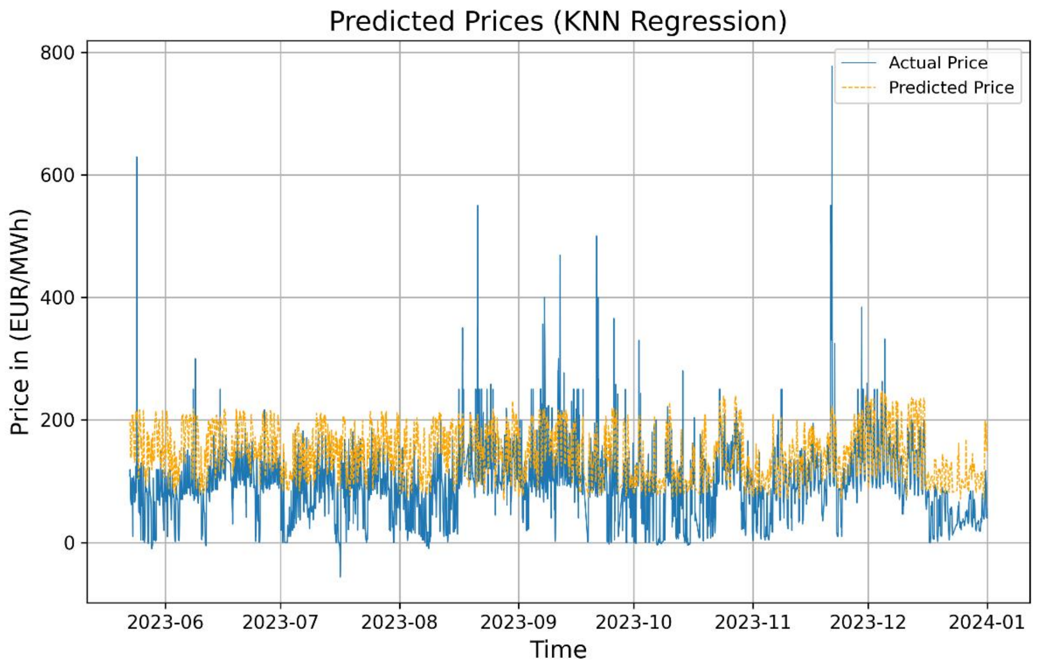

Figure 13: A graphical representation of the KNN Regression models predicted electricity price versus actual electricity price.

4.2.3. Support Vector Regression Model Results

This section presents the optimal hyperparameter configuration for the Support Vector Regression model.

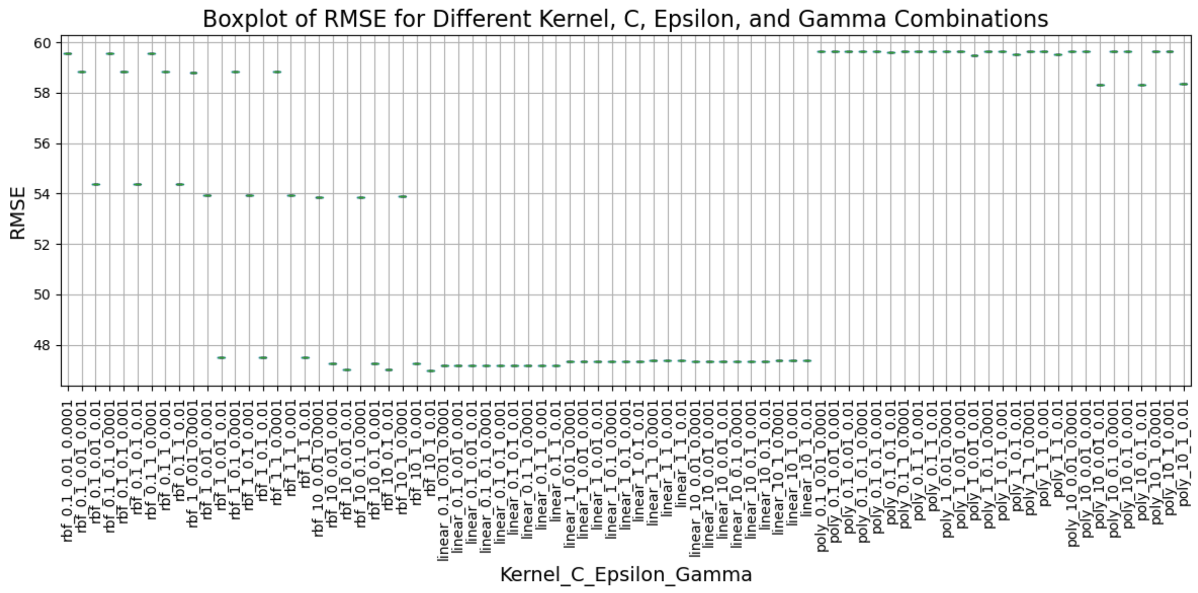

Figure 14: A graphical representation of the SVR hyperparameter combinations.

The optimal hyperparameters for SVR, as determined through grid search, are as follows:

- Kernel: rbf
- C: 1
- Epsilon: 0.01
- Gamma: 0.01

This section presents a comprehensive metric evaluation of the optimal Support Vector Regression model, accompanied by a graph illustrating the relationship between actual electricity prices and predicted electricity prices.

MSE	MAE	RMSE	R^2
2257.9933	34.2647	47.5183	0.3467

Table 4: Metric evaluation of the AVR model.

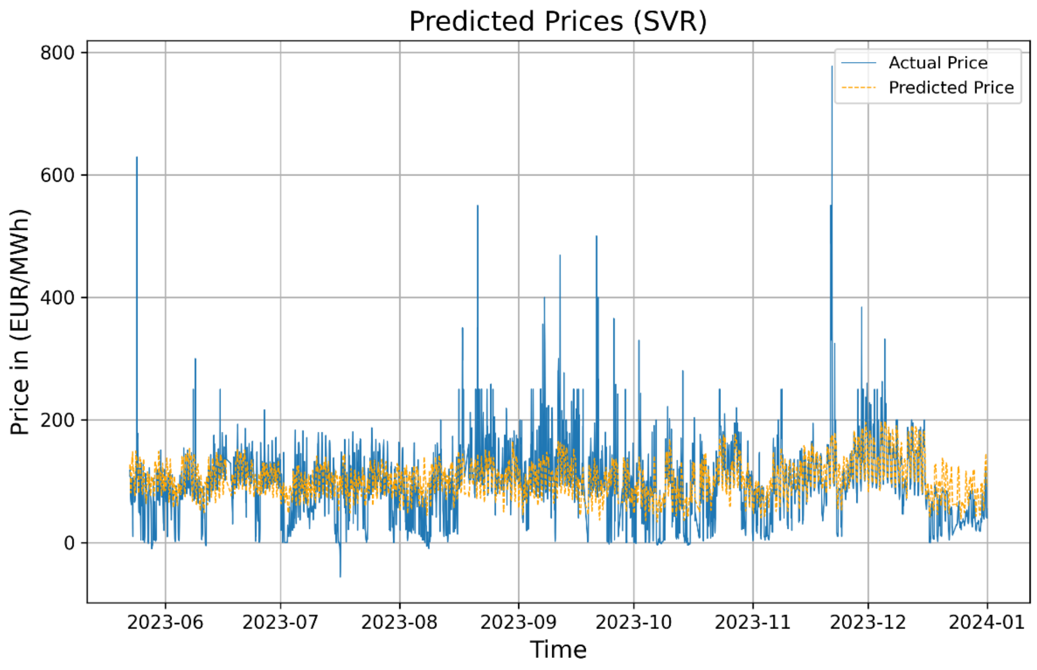

Figure 15: A graphical representation of the SVR models predicted electricity price versus actual electricity price.

4.2.4. XGBoosting Model Results

This section presents the optimal hyperparameter configurations for the Extreme Gradient Boosting model.

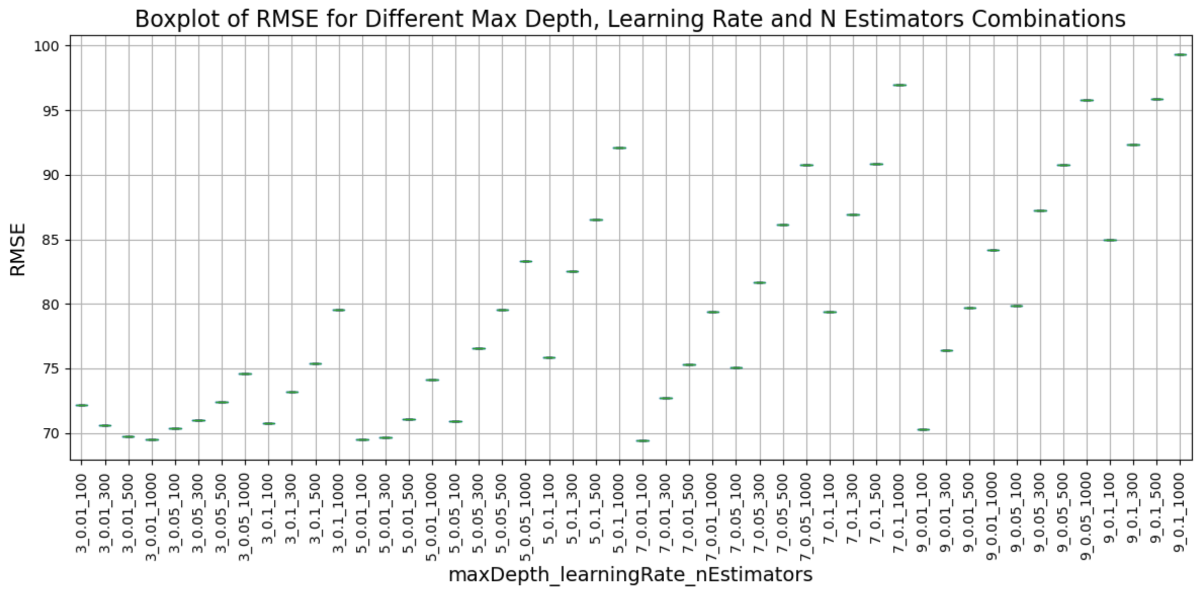

Figure 16: A graphical representation of the XGBoosting hyperparameter combinations.

The optimal hyperparameters for XGBoosting, as determined through grid search, are as follows:

- learning rate: 0.01
- max depth: 5
- n estimators: 100

This section presents a comprehensive metric evaluation of the optimal Extreme Gradient Boosting model, accompanied by a graph illustrating the relationship between actual electricity prices and predicted electricity prices.

MSE	MAE	RMSE	R^2
4915.8337	57.4384	70.1130	-0.4222

Table 5: Metric evaluation of the XGBoosting model.

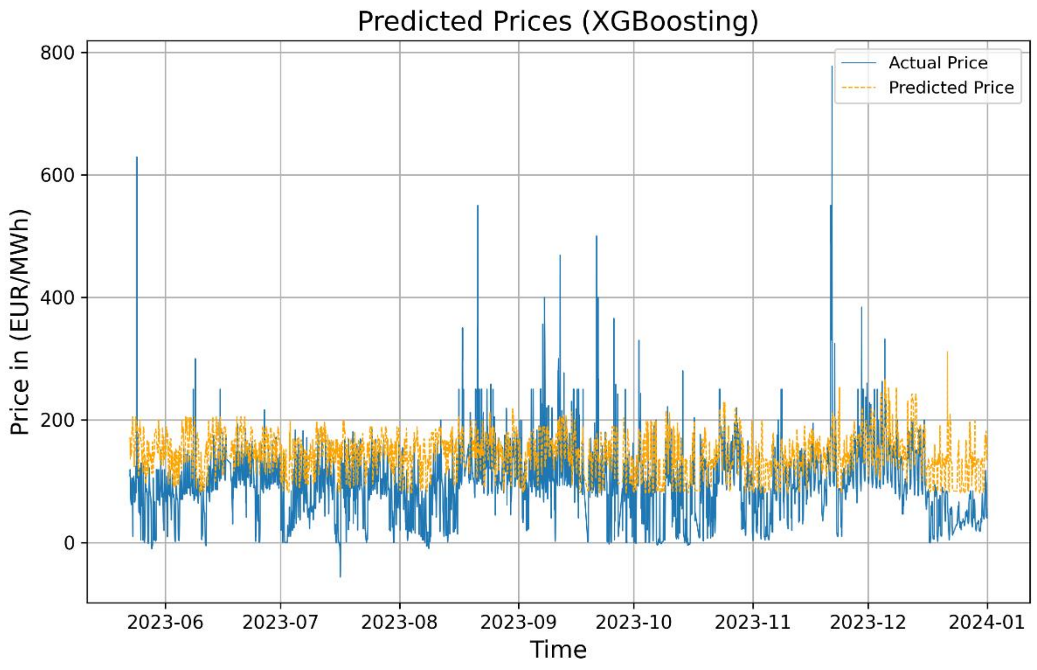

Figure 17: A graphical representation of the XGBoosting models predicted electricity price versus actual electricity price.

4.2.5. Multivariate LSTM Forecast Model Results

It is acknowledged that the outcomes of any training process will vary for artificial neural networks. Therefore, the average evaluation metrics will be presented. Subsequently, the optimal hyperparameter combination will be identified from the experiment with the lowest root mean square error (RMSE).

The output of the experiment is a graphical representation, with the y-axis representing a RMSE and the x-axis representing the hyperparameter combination. The green line represents the median performance, the blue box represents the 25th and 75th percentiles, and the whiskers represent the minimum and maximum values.

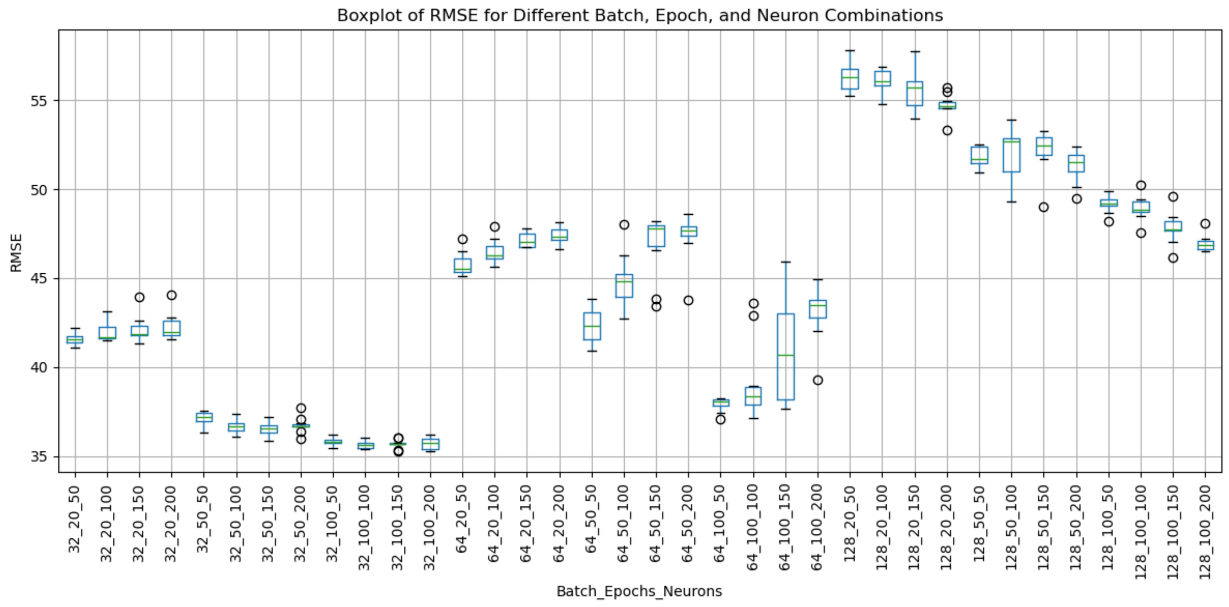

Figure 18: A graphical representation of the Multivariate LSTM hyperparameter combinations.

The optimal hyperparameters for the Multivariate LSTM model, as determined by the graph, are as follows:

- Epochs: 100
- Batch size: 32
- Neurons: 100

This section presents a comprehensive metric evaluation of the optimal Multivariate Long Short-Term Memory model, accompanied by a graph illustrating the relationship between actual electricity prices and predicted electricity prices.

Avg. MSE	Avg. MAE	Avg. RMSE	Avg. R^2
1342.1451	25.3242	36.6339	0.6117

Table 6: Metric evaluation of the Multivariate LSTM model.

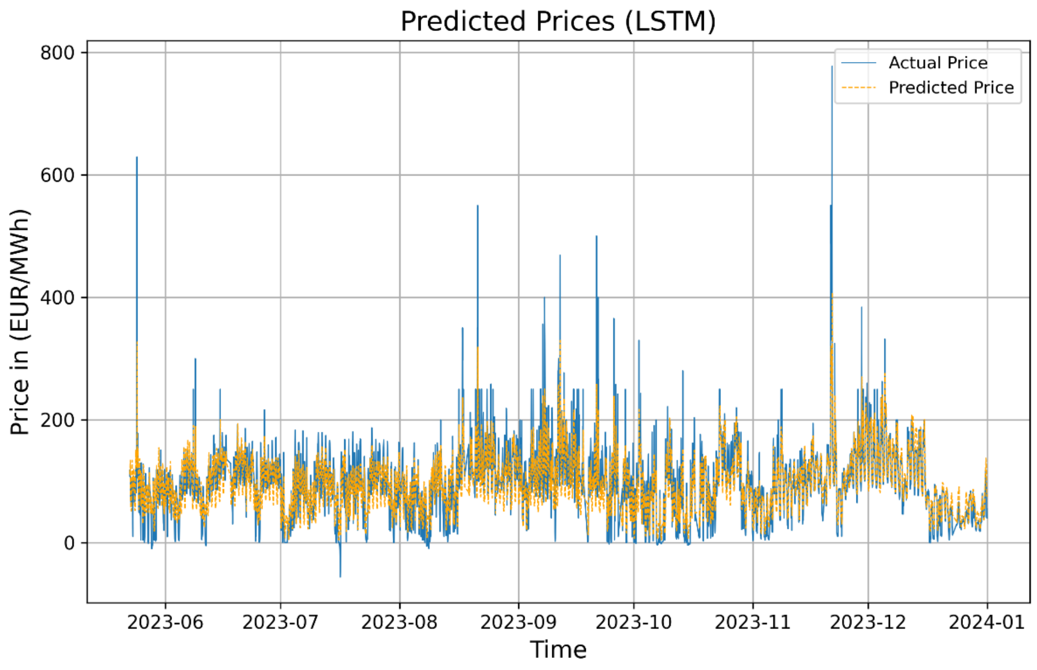

Figure 19: A graphical representation of the Multivariate LSTM models predicted electricity price versus actual electricity price.

4.3. Comparing Results

This section will present the findings of various models and their respective strengths and limitations.

Models\Metrics	MSE	MAE	RMSE	R^2
Linear regression	4364.0732	51.9762	66.0611	-0.2625
KNN regression	4835.1372	57.1536	69.5252	-0.3988
SVR	2257.9933	34.2647	47.5183	0.3467
XGBoosting	4915.8337	57.4384	70.1130	-0.4222
LSTM	1342.1451	25.3242	36.6339	0.6117

Table 7: Metric rating for all models.

Among the models assessed, the Long Short-Term Memory (LSTM) model exhibits competitive performance and stands out with the lowest Mean Squared Error (MSE) of 1342.15, indicating a smaller average squared difference between predicted and actual prices compared to other models. The LSTM model also exhibits the lowest mean absolute error (MAE) of 25.32 and root mean squared error (RMSE) of 36.63. Additionally, LSTM achieves a positive R-squared value of 0.61, indicating that it explains a substantial portion of the variance in the target variable. This suggests that the model is effective in capturing complex temporal dependencies in the electricity price data as it is designed to do so. The hyperparameter combination graph indicates that the LSTM model with a batch size of 32, an epoch size of 50 or 100, and a range of neurons from 50 to 200 performed more efficiently than other combinations.

Conversely, models such as linear regression, KNN regression, and XGBoosting demonstrate poorer performance with negative R-squared values, indicating that they perform worse than a horizontal line fit to the data. The KNN regression and XGBoosting model hyperparameter combination graph demonstrated that tuning hyperparameters does impact the efficiency of the prediction, but it did not meet the standard of having a positive R-squared value.

Notably, the support vector regression (SVR) exhibits competitive performance with a relatively low MSE of 2257.99, low MAE of 34.26, and high R-squared value of 0.35. Its performance was superior to that of linear regression, KNN regression, and XGBoosting models in terms of low MSE, MAE and positive R-squared value. Approximately half of the hyperparameter combinations depicted in the hyperparameter combination graph yielded satisfactory results in terms of RMSE.

In summary, the SVR and LSTM models have demonstrated considerable potential for forecasting electricity prices. These models exhibited superior performance across a range of evaluation metrics compared to other models.

Conclusion

In this Bachelor's project, a range of machine learning algorithms were employed, including Linear Regression, KNN Regression, Support Vector Regression (SVR), Extreme Gradient Boosting (XGBoosting) and Long Short-Term Memory (LSTM), with the objective of forecasting electricity prices in the Nord Pool market. The goal of this project was to identify the most effective algorithm for predicting electricity prices in Lithuania, taking into account external factors such as consumption, production, wind production and cross-border exchange between countries, as well as hyperparameters that impact predictions. After employing these algorithms the conclusion has been made for each of the algorithms:

1. The **LSTM model** demonstrated the most accurate predictions for Nord Pool electricity prices in Lithuania, with the lowest MAE (25.32) and RMSE (36.63), and the highest R^2 (0.61). The LSTM model is an artificial neural network that is more complex than other models and it was specifically designed for this type of problem. Therefore, the LSTM model performed as expected, yielding good results. Among all of the models, the LSTM model demonstrated the most effective ability to identify and follow up on electricity price spikes.
2. The **SVR model** demonstrated satisfactory performance with an MAE (34.26) and RMSE (47.52), yielding an R^2 (0.35). It effectively captured non-linear relationships in the data but requires careful tuning of hyperparameters and substantial computational resources. Further optimization could potentially enhance its performance.
3. The **Linear Regression model** served as a suitable baseline, with an MAE (51.98) and RMSE (66.06). However, it struggled with the complex and non-linear nature of electricity prices, resulting in a negative R^2 (-0.26). Despite its limited performance, its simplicity and interpretability make it a valuable baseline for comparison.
4. The **KNN regression model**, with an MAE (57.15) and RMSE (69.53), performed less effectively than the linear regression model, indicated by an R^2 (-0.4). This may be due to the high volume and high dimensionality of electricity price data, which the KNN regression model may struggle to handle. The selection of K=2000 may have

resulted in an over-smoothing effect. It is recommended that different features and a broader hyperparameter range be used in order to enhance the model's performance.

5. The **XGBoosting model** performed poorly, with an MAE (57.44), RMSE (70.11), and an R^2 (-0.42). Suboptimal hyperparameters likely contributed to this outcome. A more comprehensive hyperparameter tuning process, along with improved data preprocessing and feature engineering, is recommended to enhance the model's performance.

Challenges and limitations

Despite promising results, challenges are associated with data limitations and uncertainties in the electricity market. The accuracy of the model is highly dependent on the quality and completeness of the collected data. Future work may consider addressing these limitations.

Recommendations for future work

Future research could enhance forecasting capabilities by exploring more advanced machine learning techniques, such as optimal feature extraction and the inclusion of additional relevant features, as well as a greater number of hyperparameter choices. Additionally, investigating the impact of external factors, such as weather, environmental policies, or regulatory changes, could provide a more comprehensive understanding of electricity prices.

Bibliography

[AK15] Awad, M. & Khanna, R. Chapter 4 - Support Vector Regression, Journal: Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers. (2015)

[Ant23] Anthony, C. What Is Supervised Learning? (2023, January 03). Retrieved from URL: https://builtin.com/machine-learning/supervised-learning

[Baj23] Bajaj, A. Performance Metrics in Machine Learning [Complete Guide] - neptune.ai. (2023, December 8). Retrieved from URL: https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide

[Bro16] Brownlee, J. XGBoost With Python: Gradient Boosted Trees with XGBoost and scikit-learn. (2016, August 5).

[Bro20a] Brownlee, J. Multivariate Time Series Forecasting with LSTMs in Keras. (2020, October 21). Retrieved from URL: https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras

[Bro20b] Brownlee, J. How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting. (2020, August 21). Retrieved from URL: https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting

[Bro20c] Brownlee, J. A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning. (2020, August 15). MachineLearningMastery. Retrieved from URL: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning

[Bro21a] Brownlee, J. Extreme Gradient Boosting (XGBoost) Ensemble in Python. (2021, April 27). Retrieved from URL:

https://machinelearningmastery.com/extreme-gradient-boosting-ensemble-in-python

[Bro21b] Brownlee, J. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. (2021, January 13). Retrieved from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning

[Bro23a] Brownlee, J. Linear Regression for Machine Learning - MachineLearningMastery.com. (2023, August 15). Retrieved from URL: https://machinelearningmastery.com/linear-regression-for-machine-learning

[Bro23b] Brownlee, J. K-Nearest Neighbors for Machine Learning - MachineLearningMastery.com. (2020, February 24). Retrieved from URL: https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning

[BS20] N. Bakhashwa & A. Saghee. Online Tuning of Hyperparameters in Deep LSTM for Time Series Applications. International Journal of Intelligent Engineering and Systems, Vol.14, No.1. (2020, October 14). p. 212-220

[Dev23] Deval S. Top Performance Metrics in Machine Learning: A Comprehensive Guide. (2023, May 4). Retrieved from URL: https://www.v7labs.com/blog/performance-metrics-in-machine-learning

[Ger17] A. Geron. Hands-On Machine Learning with Sckit-Learn & TensorFlow. 1st. Edition. (2017, May 9).

[Gor21] Gorthy, S. Euclidean Distance - MLearning.ai. (2021, October 20). Retrieved from URL: https://medium.com/mlearning-ai/euclidean-distance-8fae145ef5f3

[GP24] Gramfort, A. & Pedregosa, F. Nearest Neighbors regression. (2024, May 12). Retrieved from URL https://scikit-learn.org/stable/auto_examples/neighbors/plot_regression.html#sphx-glr-auto-examples-neighbors-plot-regression-py

[Ibm23a] What is Machine Learning? | IBM. (Accessed 2023, November 16). Retrieved from URL: https://www.ibm.com/topics/machine-learning

[Ibm23b] What is the k-nearest neighbors algorithm? | IBM. (Accessed 2023, November 18). Retrieved from URL: https://www.ibm.com/topics/knn

[Ibm24a] IBM. What is a Neural Network? (2024, May 05). Retrieved from URL: https://www.ibm.com/topics/neural-networks

[Ibm24b] IBM. What are Recurrent Neural Networks? (2024, May 05). Retrieved from URL: https://www.ibm.com/topics/recurrent-neural-networks

[Jos18] Joseph, R. Grid Search for model tuning - Towards Data Science. (2018, December 30). Retrieved from URL: https://towardsdatascience.com/grid-search-for-model-tuning-3319b259367e

[Nem24] Single Day-ahead Coupling (SDAC). Nemo committee. (2024, May 10). Retrieved from https://www.nemo-committee.eu/sdac

[NK13] Natekin, A., & Knoll, A. Gradient boosting machines, a tutorial. (2013). Retrieved from URL: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full

[Nor23a] Day-ahead market. Nord Pool. (accesses on 2023, December 29). Retrieved from URL: https://www.nordpoolgroup.com/en/the-power-market/Day-ahead-market

[Nor23b] Intraday market. Noord Pool. (accesses on 2023, November 11). Retrieved from URL: https://www.nordpoolgroup.com/en/the-power-market/Intraday-market

[Nor24a] Find out more about Europe's leading power market. Noord Pool. (accesses on 2024, May 10). Retrieved from URL: https://www.nordpoolgroup.com/en/About-us

[Nor24b] See outline of our power market history. Noord Pool. (2024, May 10). Retrieved from URL: https://www.nordpoolgroup.com/en/About-us/History

[Ola24] Olah C. Understanding LSTM Networks. (2024, January 10). Retrieved from URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs

[Sah21] Sahani, G. R. Euclidean and Manhattan distance metrics in Machine Learning. Medium. (2020, July 24). Retrieved from URL: https://medium.com/analytics-vidhya/euclidean-and-manhattan-distance-metrics-in-machine-learning-a5942a8c9f2f

[TTM23] T. Timbers, T. Campbell, & M. Lee. Data Science: A First Introduction (2023, December 23). Chapter 7.5 – K-nearest neighbors regression.

[Tur22] How to Decide the Perfect Distance Metric For Your Machine Learning Model. (2022, April 11). Turing Enterprises Inc. Retrieved from URL: https://www.turing.com/kb/how-to-decide-perfect-distance-metric-for-machine-learning-model

[Ver23] Verma, N. An Introduction to Support Vector Regression (SVR) in Machine Learning. Medium. (2023, November 2). Retrieved from URL: https://medium.com/@nandiniverma78988/an-introduction-to-support-vector-regression-svr-in-machine-learning-681d541a829a

[Vih23] Vihar, K. Regression in Machine Learning: What It Is and Examples of Different Models. (Accessed 2023, November 16). Retrieved from URL: https://builtin.com/data-science/regression-machine-learning

[WH23] Wei, J., & He, X. Support vector regression model with variant tolerance. (2023, June 19). 56(9-10). Retrieved from URL: https://doi.org/10.1177/00202940231180620