Exchange Prediction using LSTM

Next-day prediction of the Euronext 100 Index through Recurrent Neural Network models

Anantajit Raja

March 2024

1 Introduction

This paper seeks to conduct a comprehensive comparison of various recurrent neural network (RNN) architectures in their ability to model a one-step ahead forecast of the Euronext 100 Index points using its past values. We aim to highlight the strengths and weaknesses of each approach in capturing the complex dynamics of the index. The study will also delve into the reasons behind the performance of these RNN models compared to established benchmark models, offering insights into their predictive capabilities and potential areas for enhancement in the time series forecasting of the index.

The Euronext 100 Index is a stock market index representing the performance of the top 100 companies on the Euronext stock exchange, encompassing multiple European countries.

Company	**Country**	**Mnemo**	**Sector (ICB)**	**Weight (%)**
ASML HOLDING	NL	ASML	Technology	7.71
SHELL PLC	NL	SHELL	Energy	5.56
TOTALENERGIES	FR	TTE	Energy	4.17
SANOFI	FR	SAN	Health Care	3.18
AIRBUS	FR	AIR	Industrials	3.10
SCHNEIDER ELECTRIC	FR	SU	Industrials	2.92
AB INBEV	BE	ABI	Consumer Staples	2.85
AIR LIQUIDE	FR	AI	Basic Materials	2.59
EQUINOR	NO	EQNR	Energy	2.42
ESSILORLUXOTTICA	FR	EL	Health Care	2.31

Table 1: Top 10 Companies in the Euronext 100 Index by Weight [1]

The companies are from a variety of sectors, such as finance, industrials, consumer goods, and technology, making it a broad indicator of the economic health and investor sentiment across these major European markets. Ability to predict the Euronext Index can aid in devising hedging strategies to mitigate potential losses and provides insights into the Eurozone's economic stability, essential for policymakers and businesses alike.

Machine learning models are exceptionally suited for predicting financial market indices like the Euronext 100 due to their superior pattern recognition capabilities and adaptability to the dynamic nature of markets. These models excel in identifying and learning from complex, nonlinear patterns within large volumes of historical data, without relying on traditional assumptions about market behavior. By processing and analysing extensive time-series data, machine learning algorithms can capture the intricate relationships and trends that drive market movements, offering a significant advantage over traditional financial models.

To predict the next-day movement of the Euronext 100 Index, this paper strategically employs Recurrent Neural Network (RNN) models, harnessing the specialised capabilities of Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU) models to process time series lagged features effectively. Our study on forecasting the Euronext 100 Index through RNN models tackles key challenges such as handling complex, big financial data and selecting models that best capture market dynamics. Overcoming overfitting to ensure accurate future predictions, managing the computational demands of training on extensive datasets, and addressing the unpredictability of market influencers are central to our methodology. Additionally, we prioritise model interpretability to ensure our approach is transparent and trustworthy. By succinctly addressing these hurdles, our research presents an innovative and accessible strategy for advancing financial forecasting of stock indices, such as the Euronext Index, with RNN models.

1.1 Literature Review

Our selection of models is informed by foundational works in the field. In the literature, (2017) Huynh, Dang, and Duong's [4] exploration of Bidirectional GRU models demonstrates their proficiency in handling the non-linear dynamics of financial markets by leveraging online financial news and historical stock data. This insight underscores the potential of GRU models in capturing market volatility and trend reversals, integral for predicting the Euronext 100 Index. (2020) Muncharaz [8] provides a comparative analysis that places LSTM networks above traditional forecasting models like ARIMA (Autoregressive Integrated Moving Average) in terms of reducing prediction errors for stock indices, highlighting LSTM's capability to understand and predict complex patterns over time. Further supporting the application of LSTM in financial forecasting, (2020) Moghar and Hamiche [6] illustrate the model's ability to accurately predict stock prices through learning from past data, showcasing the practical benefits of using LSTM for our application.

The inclusion of a broader perspective on neural networks' applicability in financial forecasting is crucial: (2022) Sako, Mpinda, and Rodrigues [9] offer a comprehensive review of neural network architectures, providing insights into their strengths and limitations in financial time series forecasting. This comprehensive view aids in tailoring the model selection process to the unique characteristics of the Euronext 100 Index data. Ampountolas (2023) [2] further enriches the discussion by comparing various forecasting models, including ARIMA, hybrid ETS-ANN (error, trend, seasonality) and kNN (k-nearest neighbours), across different financial markets. This study's findings on model performance during variable market conditions emphasise the importance of adaptable and robust forecasting methods, guiding our choice towards LSTM and GRU models for their proven efficacy in similar environments.

Incorporating these pivotal studies, this paper aims to develop nuanced forecasting models that integrate the advanced capabilities of RNNs, and compare them with more well-known benchmark models. The emphasis on time series lagged features aligns with the broader objective to develop a predictive model that is both sensitive to the underlying patterns in past index movements and robust against market volatility, without the need for external data inputs.

2 Methodology

2.1 Data Cleaning, Analysis and Visualisation

For our analyses we source data from the Yahoo! Finance website, leveraging 5 years of historical data spanning from 20th February 2019 to 19th February 2024. The data comprises several metrics for tracking the daily performance of the Euronext 100 Index including Open, High, Low, Close, Volume and Adjusted Close values. However, for the purposes of our analysis, we specifically concentrate on the the Adjusted Close values. The rationale for focusing solely on this lies in its ability to reflect the index's value after accounting for factors such as dividends and stock splits, thereby providing a more accurate representation of the index's true value over time.

Upon initial examination of the dataset, there are no duplicate values, but we discover six missing values (one for each column), all occurring on Christmas Day 2019. This global holiday results in closure of financial markets and hence absence of trading data on this day (though this data isn't missing for later years, possibly due to a change in data collection/reporting practice for the holiday). Focusing on the column with adjusted closing values, there is only one that's missing, which we remove from the dataset given its minimal impact on our analyses. We are left with 1282 data points in total for the five years.

To get a better understanding of the data we're dealing with, we produce a plot of the daily adjusted closing index values and, using pandas' describe function, obtain descriptive statistics for the 5 year time span, resulting in the plot in **Figure 1** and table in **Figure 2**.

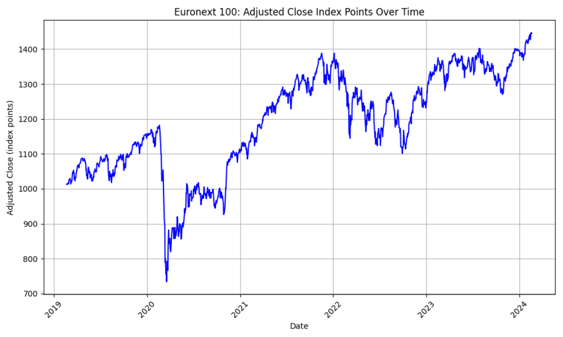

Figure 1: Plot of Euronext 100 Index points (adjusted close) from 20/02/19 to 19/02/24

	Adj Close (pts)
count	1282
mean	1190.76
std	145.68
min	733.93
25%	1082.66
50%	1212.68
75%	1318.24
max	1447.13

Figure 2: Summary statistics of index points (adjusted close) from 20/02/19 to 19/02/24

The Euronext 100 Index's journey over the observed period begins with an initial uptrend, reflecting a period of economic growth and investor confidence, as evidenced by the rising Adjusted Close points. However, this growth trajectory was abruptly disrupted by the onset of the COVID-19 pandemic, which led to a significant market downturn. The pandemic's impact is starkly evident in the data, with the Adjusted Close points dropping to a low of around 734 points during the height of market uncertainty and fear regarding the global economic outlook. Despite this sharp decline, the index demonstrated resilience and a capacity for recovery in the subsequent period. Post-pandemic, the index has been on a generally upward trend, indicating a gradual recovery and adaptation to the new economic conditions, investor optimism, and the reopening of global economies. On average, the Adjusted Close points stand at around 1190, with a standard deviation of around 145 points, reflecting the volatility and fluctuations within this period. The maximum value reached is around 1447 points, showcasing the index's ability to not only recover but also achieve new highs in the wake of the pandemic's challenges.

The variance observed in the market emphasises the importance of selecting models capable of capturing sophisticated relationships in financial time series data. RNN models excel in learning from sequential data, making them well-suited for predicting future movements of the Euronext 100 Index.

2.2 Defining the neural network models

We must understand the relevant models that can be used to effectively apply them. Feedforward neural networks process inputs independently, without considering the order in which data points are received. This is a significant limitation for financial time series forecasting, where the relationship and order of data points over time are crucial. We therefore consider using recurrent neural network models instead for our stock index forecast: in contrast to feedforward neural networks that don't have loops and where information flows unidirectionally, RNNs present cycles of neuronal connections in which a neuron can receive feedback from its output:

Definition 1. *(Recurrent Neural Networks) A Recurrent Neural Network (RNN) is a type of neural network designed for processing sequential data, incorporating cycles in its connections to retain a form of memory. This memory allows it to use information from previous inputs to influence the current processing, making RNNs suitable for tasks involving time-dependent sequences for prediction.*

RNNs are called "recurrent" because they perform the same task for every element of a sequence, with the output being dependent on the previous computations.

Mathematically, for each timestep t the hidden state h_t is updated through a combination of the current input x_t and the previous hidden state h_{t-1}, typically using a non-linear activation function ϕ (typically the hyperbolic tangent function: $\phi(x) = \tanh(x)$):

$$h_t = \phi(Wx_t + Uh_{t-1} + b_h)$$

where b_h is the bias vector for the hidden state and W and U are the weight matrices (transforming input data to the hidden state, and previous hidden state to current hidden state respectively). This represents a single vanilla unit. The hidden state at time t can then be used to produce an output $\hat{y}_t$ through another activation function ψ (typically a linear function, so ψ is usually omitted from the following equation, but we include it for completeness):

$$\hat{y}_t = \psi(Vh_t + b_y)$$

where b_y is the bias vector for the output and V is the weight matrix for the hidden-to-output connection.

The process of improving the RNN involves adjusting its inner workings based on the total loss. This is where backpropagation through time (BPTT) comes in:

Definition 2. *(Backpropagation through time) BPTT is a technique for training Recurrent Neural Networks by "unrolling" them across time steps, transforming them into a deep feed-forward network where each layer corresponds to a time step. It then applies standard backpropagation on this structure to update weights, factoring in the entire sequence's contributions to accurately capture temporal dependencies.*

For simple RNNs, the vanishing gradient problem emerges as a critical challenge during the training phase. This issue is rooted in the mechanism of BPTT - the gradients become exponentially smaller as they travel backward through the timesteps (multiplication of terms less than one), leading to gradients that approach zero, resulting in negligible weight updates. This phenomenon worsens the network's ability to learn from data points early in the sequence (their influence on the output diminishes) making long-term dependencies difficult to capture. Alternatively we may have an exploding gradient problem where gradients may become exponentially large.

To combat these potential problems, we introduce advanced architectures such as Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRUs). These incorporate mechanisms that allow the network to selectively remember and forget information, making it possible to maintain a more stable gradient flow across many timesteps:

- LSTMs have three gates that control the flow of information: 1. Forget gate, which decides what information is discarded from the cell state. 2. Input gate, which determines which values in the cell state are updated. 3. Output gate, which decides what the next hidden state should be (based on the cell state), and determines the output of the LSTM cell at the current time step.

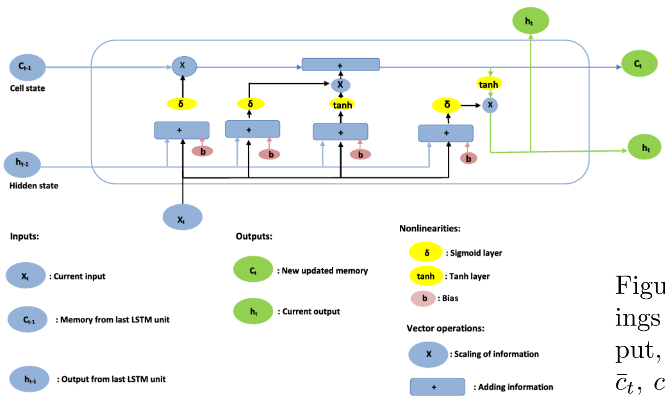

Figure 3: Inner workings of an LSTM cell [7]

$$\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}$$

Figure 4: Equations that describe the internal workings of an LSTM cell where f_t, i_t, o_t are the forget, input, output gates' activation vectors respectively, and $\bar{c}_t$, c_t, h_t are the cell candidate, cell state and hidden state vectors respectively. W and U are weight matrices for the input and hidden state, σ and b are the activation functions and bias vectors respectively; $\odot$ representing the dot product

These gates use different weight matrices and have their own activation functions, usually sigmoid for the gates themselves ($\sigma_g(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$) and tanh for creating the new cell candidate and determining the final output of the LSTM cell ($\sigma_c(x) = \sigma_h(x) = \tanh(x)$).

- GRUs simplify the LSTM model with two gates: 1. Update gate which decides how much of the past information needs to be passed along to the future (combines 'input' and 'forget' from LSTM). 2. Reset gate which determines how much past information to forget.

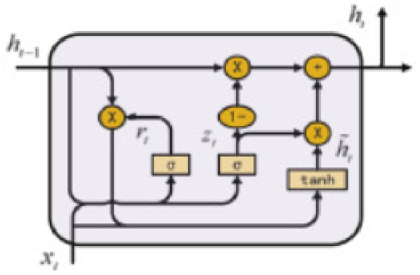

Figure 5: Inner workings of a GRU cell [3]

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

Figure 6: Equations that describe the internal workings of a GRU cell where z_t, r_t are the update and reset gates' activation vectors respectively, and $\hat{h}_t$ and h_t are the candidate activation and hidden state vectors respectively. W and U are weight matrices for the input and hidden state, σ and ϕ are the activation functions and b the bias vectors; $\odot$ representing the dot product

Again, a sigmoid activation is used for the gates themselves ($\sigma_g(x) = \text{sigmoid}(x)$), and tanh is used for updating the candidate activation $\hat{h}_t$ ($\phi_h(x) = \tanh(x)$).

These advanced RNN architectures significantly improve each network's ability to learn from long sequences by addressing the vanishing and exploding gradient problems, opening up new possibilities for applications in financial time series forecasting tasks such as ours.

Note that various other techniques, such as gradient clipping for capping gradients during backpropagration, L2 regularisation to penalise large weights, and layer normalisation, have proven effective in mitigating the vanishing and exploding gradient problems. However, our study's scope does not extend to their implementation, as we focus on the inherent capabilities of LSTM and GRU cells to address potential gradient instability within our chosen modelling framework.

2.3 Application of RNNs to stock index prediction

We now move onto applying the aforementioned architectures described to produce an accurate forecast of the Euronext 100 Index.

Prior to training the networks, we perform a feature scaling procedure to normalise both the features and the target variable. This preprocessing step is essential to ensure that the gradient descent algorithm converges more efficiently and mitigates the risk of certain features disproportionately influencing the model due to their scale. We employ the MinMaxScaler from scikit-learn's preprocessing module to scale the feature values to a bounded interval of [0, 1]. The transformation is given by $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$ where X_{min} and X_{max} are the minimum and maximum values of the feature in the training set respectively. For our main models, we will use an 80:20 train-test split, as most often used in literature.

To avoid data leakage and ensure a proper evaluation of the model's generalisation capability, the scaler was fitted only on the training dataset. The same scaler was then used to transform the test dataset. The features scaled included all numerical columns excluding 'Date' and 'Adj Close', the latter being the target variable. 'Adj Close' was also normalised using a separate MinMaxScaler, which can then be inversely transformed to the original scale post-prediction.

Our networks require input data in the form of sequences to model time dependencies. We structured our data into subsequences, each consisting of $n_{lags} = 5$ days of lagged features (which at time t is our x_t vector, mentioned earlier) to predict the current day Euronext 'Adj Close' value.

This data transformation resulted in the input features X comprising a three-dimensional array with dimensions [samples, time steps, features], suitable as input for our RNN models. The corresponding target y is the 'Adj Close' value of the current day that follows each sequence of lagged features.

Given the sequential nature of the data we use cross-validation on a rolling basis to identify the optimal number of neurons in the hidden layer. The dataset is divided into training and testing sets based on predefined split ratios (0.2, 0.4, 0.6, 0.8), progressively increasing the size of the training set, preserving order such that the test set is always after the training set (since we're interested in accuracy of future predictions).

For each combination of units and split ratio, the RNN models (simple RNN, GRU, LSTM) are trained and evaluated. The Root Mean Square Error (RMSE) metric is calculated for each model in the validation phase, providing a quantitative measure of the model's prediction accuracy on the test set.

Definition 3. *(Root Mean Square Error (RMSE)) The root mean square error can be defined mathematically as square root of the average of the sum of squared differences between the predicted and the actual values:* $RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(\hat{y}_t - y_t)^2}$ *where N is the number of observations of the time series for which a prediction is produced, y_t is the observed value at time t and $\hat{y}_t$ is the output of the model for time t.*

The results are then aggregated for each model using the average RMSE across the split ratios. We select the hidden layer 'units' hyperparameter using, for example, the plot in **Figure 7** for our LSTM model.

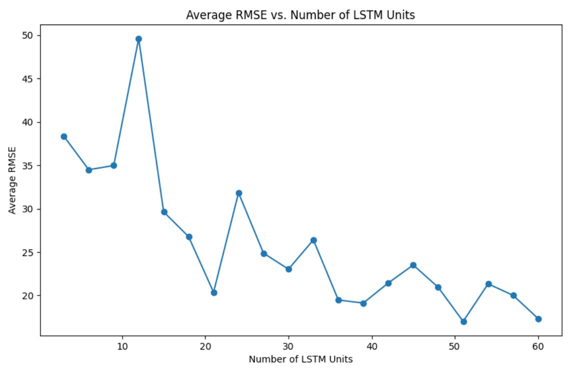

Figure 7: Plot of RMSE against number of hidden units used for the LSTM model

Various configurations of units are explored to determine the optimal model complexity. This experimentation with the number of hidden units is aimed at finding a balance between sufficient model capacity to capture complex market dynamics and avoiding overfitting to the training data - it seems here that 21 units strikes a good balance given the model is not too complex, yet it still has a low average RMSE.

By fixing the number of hidden units at 21 for all of our RNN models, we aim to assess how well each RNN architecture utilises its structural features to learn from the same amount of information. Each of the 21 units receives the same input x_t and the same previous hidden state h_{t-1} but they process the information differently because each unit has its own set of weights and biases. This allows the network to learn different characteristics of the input sequence in parallel.

For all of our RNN models we choose the tanh (hyperbolic tangent) activation function for the hidden layer due to its specific properties that align well with time-series analysis:

- Tanh outputs values in a range between -1 and 1, offering a balanced output that can effectively model the fluctuations and trends in financial time series by normalising the data flow through the network.
- Its non-linear nature allows the model to capture complex dependencies in the data
- Tanh's zero-centered output improves the efficiency of weight updates during training, contributing to faster convergence of the model.

We apply a linear activation function to produce the output prediction:

- It allows the model to predict continuous values across a wide range, which is essential for forecasting stock market indices where the future values are not restricted to a predefined scale.
- A linear activation function does not transform the output in a complex way, making the model's predictions more interpretable.

The structure is therefore as follows for each of our RNN models:

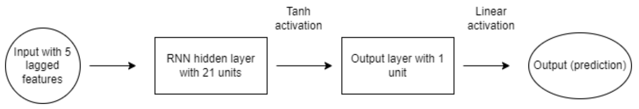

Figure 8: Illustration of how information is passed through our RNN models

We use the Adam optimiser for all of our models because it adjusts the learning rate dynamically for each weight in the model by estimating the first (mean) and second (uncentered variance) moments of the gradients, making it efficient in practice and well-suited for problems with large datasets such as our one.

We use as loss function the Mean Squared Error (MSE) because it emphasises larger errors by squaring the differences, thus pushing the model to reduce significant errors (by definition this is just the squared RMSE).

In training our models, we set the number of epochs to 50, which dictates the model will iterate through the entire training dataset 50 times, allowing sufficient opportunity to learn patterns within the data. The batch size, chosen as 32, determines the number of samples processed before the model's internal parameters are updated, balancing computational efficiency with the stability of the learning process.

We use a model selection strategy wherein we continuously seek to improve our RNN model's predictive accuracy. By comparing the current model's performance, as quantified by RMSE, to that of the best-performing model thus far, we ensure that only the most accurate model is retained. If the error of the current model is less than or equal to the lowest RMSE recorded we update to this new value and proceed to serialise the model using Python's 'pickle' module, effectively preserving the current state of the model for future use.

2.4 Defining benchmark models and applying them to stock index prediction

In forecasting, particularly for financial indices like the Euronext 100, it is crucial to establish null models to serve as benchmarks. These simple models provide a reference point against which the performance of more complex models can be evaluated. They validate that the improvements from machine learning models are statistically significant and not due to random chance, helping to contextualise the performance.

To introduce our main benchmark model we must do some further statistical analyses of the time series of index points.

Definition 4. *(Autocorrelation) Autocorrelation reflects the degree of similarity between the series and a lagged version of itself over successive time intervals. Let X_t be a time series. We calculate the autocorrelation as: $R(k) = \frac{\sum_{t=k+1}^{T}(X_t-\bar{X})(X_{t-k}-\bar{X})}{\sum_{t=1}^{T}(X_t-\bar{X})^2}$ where T is the number of time series data points.*

Definition 5. *(Partial Autocorrelation) Partial autocorrelation is a measure of the direct relationship between an observation in a time series X_t and another observation at a certain lag k away, X_{t+k} (removing the effect of lags in between). We define the partial autocorrelation function as: $\phi(1) = corr(X_{t+1}, X_t)$ when $k = 1$, $\phi(k) = corr(X_t - \hat{X}_t, X_{t+k} - \hat{X}_{t+k})$ when $k \geq 2$ where $\hat{X}_t$ and $\hat{X}_{t+k}$ are linear combinations of the smaller lags that minimise the mean squared error with respect to X_t and X_{t+k}.*

In our analysis, we visualise the autocorrelation and partial autocorrelation functions (ACF and PACF) for the Euronext 100 Index points, depicted in **Figure 9**, using data from the training set. These graphical representations are pivotal for discerning the memory characteristics of the index's movements. Specifically, the ACF plot aids in determining if the index exhibits a "long memory" trait, which is inferred from a slow reduction in autocorrelation values even across extended lag periods. This implies that historical values of the index exert a lasting impact on its future values.

The PACF plot reveals an abrupt decline in correlation following the initial lag, a signature feature of an AR(1) model (autoregressive with lag parameter equal to 1). This pattern suggests that the immediate past value predominantly shapes the current index value, with minimal effect from older historical data.

The AR(1) model is therefore suitable for use as our main benchmark for one-step predictions of the Euronext 100 Index.

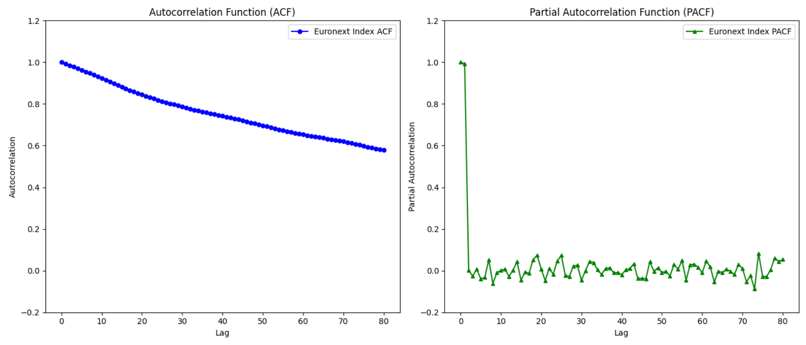

Figure 9: Plots of Euronext 100 Index autocorrelation and partial autocorrelation functions for a training set that makes up eight-tenths of the data

We also introduce three more elementary benchmark forecasting methods: naive forecast, average forecast, and weighted average forecast, which serve as foundational comparisons to evaluate the added complexity of models already introduced:

- Naive Forecast: This method simply predicts the next value in a series as the last observed value. It's the most straightforward approach, under the assumption that the most recent observation is the best predictor of the immediate future.
- Average Forecast: We predict the next value by calculating the average of the 5 previously observed values.
- Weighted Average Forecast: This approach enhances the average forecast by assigning more weight to more recent observations and less weight to older ones, reflecting the belief that more recent data is more indicative of future values. Weights decrease linearly from the most recent observation back to the oldest one used in the forecast i.e. a weight of 5 is used for the most recent observation and a weight of 1 is used for the least recent.

For each method, predictions are initially made on a scaled dataset to align with preprocessing steps of complex models, and then inversely transformed to their original scale for a fair comparison.

To rigorously assess the performance of our models, we utilise a trio of established evaluation metrics: Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and RMSE.

Mean Absolute Error (MAE) offers a straightforward interpretation by providing the average magnitude of errors between predicted and actual values, without considering their direction. Lower MAE values indicate higher precision, making it an intuitive measure of model performance. Mean Absolute Percentage Error (MAPE) expresses the average absolute error as a percentage of actual values, allowing for relative comparison of models across different scales. And, as we've seen previously, RMSE is a quadratic scoring rule that assesses the average magnitude of the error.

By evaluating our models against these metrics, we aim to capture a well-rounded understanding of their predictive performance, enabling us to identify the model that most effectively balances accuracy with error sensitivity in the context of the one-step ahead Euronext 100 Index forecast.

3 Results

In this section we present a comprehensive comparison of the predictive performances of the various forecasting models applied to the Euronext 100 Index. Our analysis spans both sophisticated deep learning models (including simple RNN, GRU, and LSTM networks) and traditional benchmark models (namely the AR(1), naive forecast, average forecast, and weighted average forecast models).

We first evaluate the models using the MAE, MAPE and RMSE error metrics, obtaining the results in Table 2.

Model	MAE	RMSE	MAPE
LSTM	4.96	6.34	0.37
Simple RNN	2.12	2.63	0.15
GRU	3.99	5.29	0.30
AR(1)	7.81	10.29	0.58
Naive Forecast	7.79	10.32	0.58
5-day Average	11.24	14.73	0.83
5-day Weighted Average	9.85	12.91	0.73

Table 2: Comparison of forecast models using MAE, RMSE, and MAPE.

The LSTM model showcased a competitive edge, yet was outperformed by the simple RNN and GRU models in these metrics. The simple RNN model reported the lowest values across all three metrics, indicating an impressive balance between complexity and performance. Simple RNN's more streamlined architecture is more efficient at capturing the necessary patterns without the additional mechanisms that LSTMs/GRUs provide for longer-term dependencies. LSTMs excel in scenarios where long-term dependencies are crucial, which does not seem to be the case here. Since our lookback period is only 5 days, the complexity of the LSTM and GRU models may have lead to slight overfitting, where the models learn the noise in the training data instead of the underlying trend.

The traditional AR(1) model, while outstripped by the recurrent neural networks, still delivered a reasonable performance, outdoing both average forecasts. The naive forecast yielded MAPE and RMSE values that, while not minimal, suggest its usefulness as a baseline model. It seems that models which put the most emphasis on the last observed value give better next-day forecasts as expected, underscoring the relevance of more recent data points in predicting the index. Supporting this, the weighted average forecast performed better than the simple average forecast across all metrics. However, it still fell short compared to the more advanced models.

Aside from the application of error metrics for analysing the statistical validity of our forecasting results, we have also ensured robustness of these results through the meticulous approach to time series cross-validation defined earlier in the methodology. It allows most observations to have the opportunity to be in the test set - the approach helps in mitigating the variance of the model's performance metrics, as it reduces the likelihood of overfitting to a particular subset of data, ensuring that the model's effectiveness is not contingent on the specificities of a single test set.

Fold	**Training Data Range**	**Test Data Range**
1	0 - 0.2	0.2 - 0.4
2	0 - 0.4	0.4 - 0.6
3	0 - 0.6	0.6 - 0.8
4	0 - 0.8	0.8 - 1

Table 3: Cross-Validation Scheme

4 Discussion

We can also visualise our findings by overlaying the actual test set values (say the final 20% of the entire dataset) with our model's predictions. We gain a more intuitive and comparative perspective of how closely each model's forecasts align with reality - see **Figures 10, 11 and 12**.

Note that the naive forecast isn't exactly a translated version of the graph of actual values due to there being no trading activity on weekends and on holidays. This results in blips and slight deviations from the shape of the graph of the actual values because the next business day is not always the next calendar day.

We can see how the average and weighted average forecasts dilute the influence of most recent short-term trends (**Figure 10**), smoothing over significant fluctuations, leading to larger deviations from actual values observed. In contrast, the AR(1) and naive models, by focusing more closely on the most recent observation or incorporating it with a lagged dependency respectively, are better aligned with the market's short-term momentum, allowing them to closely mirror actual values with their relatively smaller deviations.

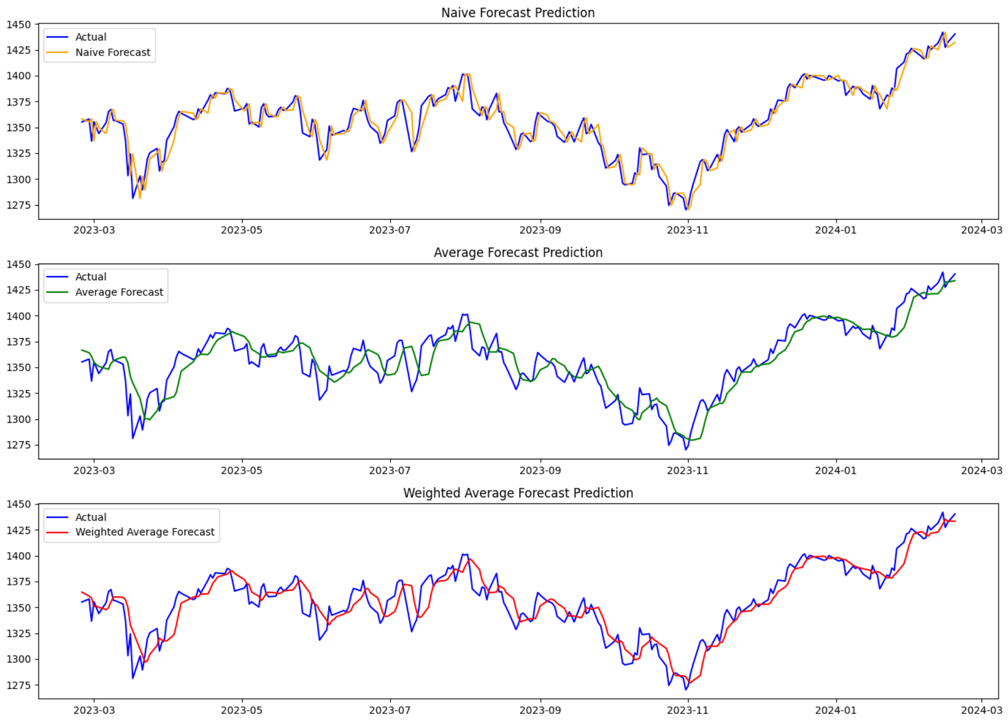

Figure 10: Plots of predictions of naive, average and weighted average models vs actual values for one-step ahead forecast of Euronext 100 on the test set

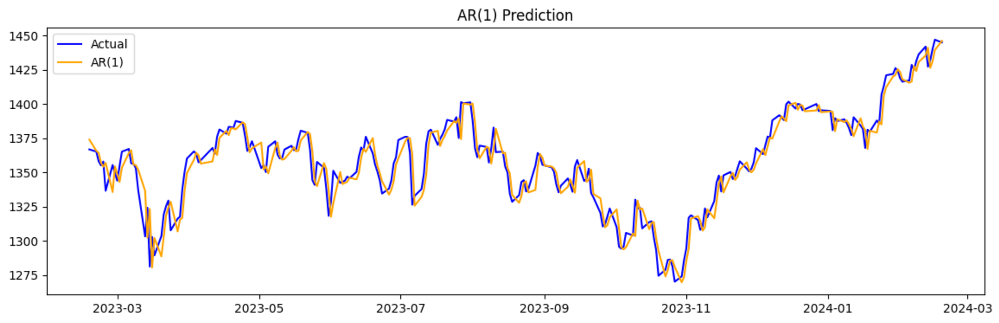

Figure 11: Plot of predictions of AR(1) model vs actual values for one-step ahead forecast of Euronext 100 on the test set

The seemingly remarkable similarity between the forecasted values and the actual values observed in the plots can be attributed to the nature of the forecasting task itself, which focuses on predicting the value for the next day based on information available up to the current day. The value of the index on any given day is typically not drastically different from the previous day's closing value.

While the benchmarks offer a fundamental perspective on the time series, the RNNs capture the temporal dynamics of the data better due to their recursive nature, as reflected in their more closely aligned forecast plots with the actual data (**Figure 12**).

The nonlinearity ingrained in these models via activation functions enables them to encapsulate complex and irregular patterns found in factors influencing the Euronext 100 stocks. The notorious vanishing and exploding gradient problems are less of an issue here with due to the short-term forecasting horizon and limited lookback period. Each network only needs to backpropagate through a small number of time steps. This short sequence length mitigates the risk of gradients vanishing or exploding as there are fewer multiplicative

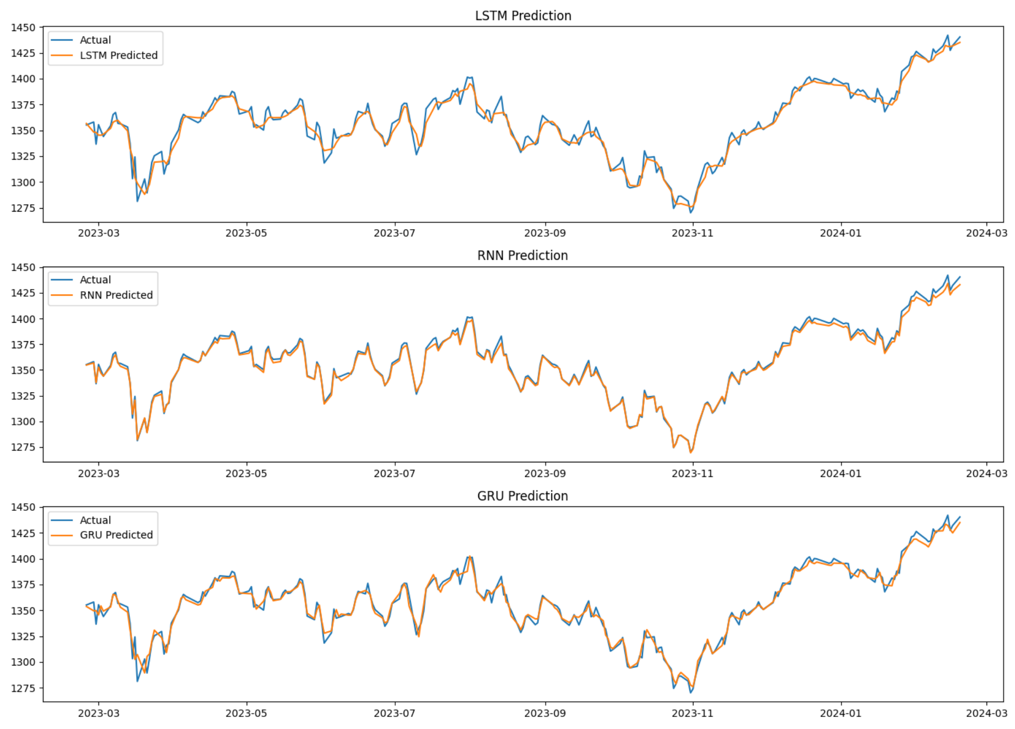

Figure 12: Plots of predictions of RNN models vs actual values for one-step ahead forecast of Euronext 100 on the test set

steps through time.

In RNNs, the concept of parameter sharing is grounded in the idea that the same set of weights is utilised across all time steps during the sequential processing of data. This reflects an underlying assumption that the process governing the sequence is invariant over time; that is, the way in which the present influences the future remains consistent, regardless of the specific moment in the sequence. This means that the statistical dependencies between consecutive data points are presumed to be stable throughout the time series, allowing the same calculations to be applied at each step to predict the next value in the sequence.

Our AR(1) benchmark model assumes that the current value of the index points time series is linearly dependent on its immediate previous value, plus a stochastic term. The model presumes stationarity, implying the time series' mean, variance, and autocorrelation structure do not change over time (though can be evidenced by an Augmented Dickey-Fuller test) [5].

The simple average forecast assumes that all past values of the index contribute equally to future values, which may not account for potential changes in trend or seasonality, whilst the weighted average forecast assumes that more recent observations are more relevant to future values, with this relevance diminishing for older data points.

The simplistic approach of the benchmark models yield quick predictions. However, this simplicity comes at the cost of expressiveness, as these models lack the structural complexity to capture deeper financial dynamics and can only reflect surface-level trends.

4.1 Perspectives for future studies

In advancing financial forecasting for the Euronext index, incorporating additional features beyond 'Adj Close', such as volume, open, high, and low values, or even exogenous variables like economic indicators or sentiment analysis from financial news, could enrich the model's input and potentially improve predictive performance.

Furthermore, advanced regularization techniques, like dropout or L1/L2 regularization, could be systematically explored to prevent overfitting, especially when increasing model complexity. The integration of hybrid models that combine RNNs with other machine learning techniques also emerges as a promising strategy to boost predictive accuracy. Innovative architectures like the transformer, which departs from traditional sequential processing, could also further refine our forecast of the Euronext 100 Index.

Prioritising the explainability and interpretability of these models is also crucial, ensuring that their forecasts and the decision-making processes they inform are transparent and credible to financial analysts and stakeholders.

Our research into forecasting the Euronext 100 Index represents a significant stride toward understanding and predicting its day-to-day movements, leveraging advanced machine learning techniques for enhanced accuracy. This work not only contributes to the existing body of knowledge by showcasing the superiority of RNN models in certain forecasting scenarios, but also highlights the importance of methodical validation and the exploration of model configurations. By addressing the nuances of this financial data through sophisticated modeling, this study paves the way for more informed decision-making in finance, offering valuable insights for investors, analysts, and policymakers aiming to navigate the complexities of the Euronext 100 Index.

References

[1] Euronext 100 index composition. https://live.euronext.com/en/product/indices/FR0003502079-XPAR/market-information. Accessed: 2024-03-15.

[2] A. Ampountolas. Comparative analysis of machine learning, hybrid, and deep learning forecasting models: Evidence from european financial markets and bitcoins. *Forecasting*, 5:472–486, 2023.

[3] Jinglong Chen, Hongjie Jing, Yuanhong Chang, and Qian Liu. Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process. *Reliability Engineering & System Safety*, 185:372–382, 2019.

[4] Huy D. Huynh, L. Minh Dang, and Duc Duong. A new model for stock price movements prediction using deep neural network. In *Proceedings of the 8th International Symposium on Information and Communication Technology*, SoICT '17', pages 57–62, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Deniz Ilalan and Özgür Özel. Unit root testing in the presence of mean reverting jumps: Evidence from us t-bond yields. *International Journal of Nonlinear Sciences and Numerical Simulation*, 20(2):145–152, 2019.

[6] Adil Moghar and Mhamed Hamiche. Stock market prediction using lstm recurrent neural network. *Procedia Computer Science*, 170:1168–1173, 2020.

[7] M. Mroua and A. Lamine. Financial time series prediction under covid-19 pandemic crisis with long short-term memory (lstm) network. *Humanities and Social Sciences Communications*, 10:530, 2023.

[8] Javier Muncharaz. Comparing classic time series models and the lstm recurrent neural network: An application to sp 500 stocks. *Finance, Markets and Valuation*, 6:137–148, 01 2020.

[9] K Sako, BN Mpinda, and PC Rodrigues. Neural networks for financial time series forecasting. *Entropy*, 24(5):657, 2022.