University of Tartu
Faculty of Science and Technology
Institute of Mathematics and Statistics

William Vaask

Comparative Analysis of Traditional Time Series, Machine Learning, Deep Learning and Hybrid Models for Profit Forecasting in Financial Markets

Actuarial and Financial Engineering

Master's Thesis (30 ECTS)

Supervisor: Prof. Toomas Raus

Tartu 2025

AEGRIDADE LINEAARSETE, MASINÕPPE, SÜVAÕPPE JA HÜBRIID MUDELITE VÕRDLEV ANALÜÜS KASUMI PROGNOOSIMISEL FINANTSTURGUDEL

Magistritöö

William Vaask

Lühikokkuvõte

Töös võrreldi lineaarsete, masinõppe, sügavõppe ja hübriid mudelite aegridade prognoositäpsust panganduses finantsturgude valdkonnas, prognoosides kolmel erineval tasemel agregeeritud igapäevast kasumit. Erinevate meetodite võrdlemiseks kasutati uut jõudlusmõõdikut - korrigeeritud keskmine skaleeritud absoluutviga (cMASE), mis parandab MASE tõlgendatavust, kasutades T ühe-sammu naiivset prognoosi $T-1$ asemel, mille tulemusel on naiivse meetodi skoor alati $cMASE = 1$.

Vaatamata arvutusvõimsuse arengust tingitud võimalikele mudelitele, oli lineaarne aegreamudel SARIMA teistest mudelitest parem, näidates kõige järjepidevamaid tulemusi keskmise ristvalideerimise cMASE ja testimise cMASE vahel erinevate aegridadega.

Parimates hübriidmudelites täiendasid gradiendi võimenduse meetodid SARIMA-t, parandades prognoose ajaliselt pikkade nihete, liikuvate statistikute abil. Kui SARIMA mudelid vajasid uuesti treenimist peale igat prognoosi, siis masinõppe, süvaõppe ja hübriid mudelite mittelineaarsed osad prognoosisid kõige paremini, kui neid treeniti uuesti ainult keskmiselt iga kahe nädala tagant, mis vähendas oluliselt kogu arvutuskulu.

CERCS teaduseriala: P160 Statistika, operatsioonianalüüs, programmeerimine, finants- ja kindlustusmatemaatika.

Märksõnad: MASE, SARIMA, XGBoost, LightGBM, LSTM, Hübriid.

COMPARATIVE ANALYSIS OF TRADITIONAL TIME SERIES, MACHINE LEARNING, DEEP LEARNING AND HYBRID MODELS FOR PROFIT FORECASTING IN FINANCIAL MARKETS

Master thesis

William Vaask

Abstract

This thesis compared the forecasting performance of traditional time series, machine learning, deep learning and hybrid models on daily banking profit data in financial markets area aggregated on three different levels. To evaluate different methods, this thesis used a novel performance metric - corrected mean average scaled error (cMASE), which improves interpretability of MASE by using T one-step naive forecasts instead of $T-1$, which results in naive method always having a score of $cMASE = 1$.

Despite various models available due to advancements in computational power, traditional time series method SARIMA still outperformed other models, showing the most consistent results between average cross-validation cMASE and testing cMASE.

For best hybrid models, gradient boosting complemented SARIMA by correcting forecasts using long lags and rolling statistics. While SARIMA models required refitting after every forecast, machine learning, deep learning and non-linear parts of hybrid models performed best when refit only on average once every two weeks, which reduced the overall computing cost significantly.

CERCS research specialisation: P160 Statistics, operations research, programming, financial and actuarial mathematics.

Key Words: MASE, SARIMA, XGBoost, LightGBM, LSTM, Hybrid.

Contents

Introduction

Forecasting profits has been important for business planning and business strategy for a long time. Accurate forecasts enable banks to allocate resources more effectively and optimize their operations. Traditionally, statistical methods like ARIMA and SARIMA have been used for forecasting, but advancements in computational power have introduced techniques such as machine learning and deep learning, which could lead to superior accuracy and adaptability in varying market conditions.

Although there are numerous forecasting methods nowadays, there is no clear consensus on which approach consistently performs best under varying market conditions. As it is almost universally accepted that there is no one forecasting method that is better than all others for every forecasting situation (Kontopoulou et al., 2023), this study focuses on forecasting profit for banking in financial markets area. Furthermore, this is important as there are currently no clear best methods for different types of market data forecasting challenges. This creates challenges for different practitioners who must choose models without clear guidance on their relative strengths and weaknesses. This thesis attempts to find best methods for forecasting profits comparing methods on three levels of complexity. This research also aims to find the overall best method for forecasting profits in the market.

To achieve this, the primary objective of this study is to compare the performance of traditional time series models to machine learning methods, deep learning techniques, and a hybrid method for profit forecasting. The research aims to provide clarity to banks and other businesses using market data in their day-to-day operations by analyzing how these models perform with different complexity of time series data across multiple metrics.

1 Instruments in banking in financial markets area

In capital markets, banks and other financial institutions often trade with many different securities. Some of the most common securities that all large banking institutions deal with in their day to day operations are the following: fixed income, equities, foreign-exchange, credit and interest-rate derivatives (Liaw, 2011). In financial market area the most complex units that deal with these instruments are trading desks. There are several different possible trading desks in investment and commercial banks, for example fixed income desks, equity desks, foreign exchange desks, sales desks or commodities desks (Hull, 2018). Each of these desks usually compromises of several portfolios that are managed by the traders from that trading desk which trade with various financial instruments, often taking large hedged positions as a trading strategy to manage the risk generated by any trade (Wosnitzer, 2016, Lee, 2019, Anand et al., 2012). Due to this it is often more informative not to only look at profit from an instrument but its hedged positions as well.

For investment portfolios, banks tend to prefer more illiquid assets like fixed-income assets with modest fundamental risk (Hanson et al., 2015). Fixed income securities are debt instruments, which are issued by borrowers to make certain promised stream-of-cash flows in future (Sundaresan, 2009). Promised stream-of-cash flows may be secured by specific assets of the borrowers or they can be unsecured (Sundaresan, 2009).

According to the Capital Asset Pricing Model (Lintner, 1965; Sharpe, 1964) investors are not compensated for risks that can be diversified. Uncompensated risk is the market return. According to this, there are two efficient assets, an almost risk free asset with minimal return and a market asset

which is more compensating than risk free asset due to enhanced risk. One of the most common risk free assets is government bonds (Yankov, 2014). Government bonds are often used as time value of money, as the value of bonds are based on discounted value of the future payments received, so bonds characterize money in the future being worth less than at present time (Van Binsbergen, Diamond, and Grotteria, 2022).

While banks do hold assets like equities, traditional banking does not hold equities with risky cash flows (Hanson et al., 2015). Equity securities represent ownership claims on an entity, for example a company.

A foreign exchange (FX) derivative is a financial derivative whose payoff depends on the FX rates of two or more currencies (Kallianiotis, 2013). These instruments are commonly used for hedging purposes (Kallianiotis, 2013).

A credit derivative is an agreement designed to shift default risk from one party to the other (Mengle, 2007). Credit derivative's value is derived from the credit performance of one or more corporations, sovereign entities, or debt obligations (Mengle, 2007).

Interest rate derivatives are often used to manage interest rate risks (Brewer III, Minton, and Moser, 2000). Interest rate risk management improves intermediation efficiency of banks as it allows banks to take on more credit risk (Diamond, 1984; Brewer III, Minton, and Moser, 2000). Some of the most commonly used interest rate derivatives are forward rate agreements (FRA), swaps, swaptions and futures as they are most widely used in banking (Flavell, 2010). FRA is an agreement between two counterparties, where the seller agrees to pay a floating rate interest and receive a fixed rate interest, whereas buyer agrees to pay a fixed rate interest to receive a floating rate interest (Flavell, 2010). Swap is an agreement between two counterparties to

exchange the cash flows or liabilities of two different financial instruments (Flavell, 2010). Swaption is a single option on a forward swap, where buyer has an option to exercise a forward swap to receive fixed interest on a swap, while paying floating rate (Flavell, 2010). Futures are agreements to either buy or sell a financial instrument at a set future date (Flavell, 2010).

2 Models

To compare classical statistical methods to machine learning, deep learning and hybrid models, several different models are used. We will use ARIMA and SARIMA as classical statistical forecast methods. Autoregressive integrated moving average (ARIMA) and seasonal ARIMA (SARIMA) (Box and Jenkins, 1970) are the most widely used methods of classical forecasting, valued for their mathematical ability to model linear relationships.

Real word data and relationships are often too complicated to explain with linear models. In attempts to describe more complex relationships this thesis compares classical linear forecasting to nonlinear machine and deep learning methods.

For machine learning models this study uses gradient boosting methods LightGBM and XGBoost as they are widely used in financial forecasts.

It has been shown that deep learning models, particularly long short-term memory (LSTM) has shown great performance in capturing long-term dependencies in time series data (Mochurad and Dereviannyi, 2024), outperforming traditional time series models (Siami-Namini, Tavakoli, and Namin, 2018; Siami-Namini and Namin, 2018; Sirisha, Belavagi, and Attigeri, 2022). Their ability to learn from raw data without manual feature extraction is the reason why they perform well in advanced forecasting tasks (Hochreiter, 1997).

Hybrid models try to combine the strengths of traditional and deep learning methods. This study uses ARIMA-LSTM as such a hybrid model due to it performing well on financial stock data (Choi, 2018). This model uses ARIMA to capture linear trends and seasonality, while residual nonlinear data is passed onto LSTM (Zhang, 2003).

2.1 Traditional Time Series Models

Box and Jenkins, 1970; Raus, 2023

$ARIMA(p, d, q)$ process (p - order of autoregressive (AR) term, q - order of moving average (MA) term) is called the process Y_t, for which the d-th order differences $W_t = (1 - B)^d Y_t$ are causal weakly stationary $ARMA(p, q)$ process with the general form

$$\phi(B)(1 - B)^d(Y_t - E(Y_t)) = \theta(B)A_t,$$

where

Y_t - Random variable of a stochastic process Y at time t,

$B^i = Y_{t-i}$ - Lag operator,

$\phi(B) = 1 - \sum_{i=1}^{p} \phi_i B^i$ - Polynomial of AR term,

$\theta(B) = 1 + \sum_{j=1}^{q} \theta_j B^j$ - Polynomial of MA term,

A_t - White noise process (uncorrelated random variables with mean 0 and finite variance σ^2).

Y_t is a seasonal $ARIMA(p, d, q)x(P, D, Q)_s$ process (p, d, q are parameters of non-seasonal components, while P, D, Q are parameters of seasonal components) with period s if d and D are nonnegative integers and the differenced series $W_t = (1 - B)^d(1 - B^s)^D Y_t$ is a causal ARMA process defined

$$\phi(B)\Phi(B^s)W_t = \theta(B)\Theta(B^s)A_t,$$

where

$\Phi(B^s) = 1 - \sum_{i=1}^{p} \phi_i B^{is}$ - Polynomial of seasonal AR term,

$\Theta(B^s) = 1 + \sum_{j=1}^{q} \theta_j B^{js}$ - Polynomial of seasonal MA term,

ARMA process is causal if it can be expressed as a weighted sum of present and past error terms.

These models are influenced by specificity of the data and seasonal patterns (Szostek et al., 2024). Furthermore, this limits to characterize nonlinear dependencies and complex seasonality (Cheng et al., 2015). Due to this and advancements in computational power and algorithms researchers have begun to explore either alternative methods (Alim et al., 2020; Kontopoulou et al., 2023; Sirisha, Belavagi, and Attigeri, 2022) or hybrid methods such as ARIMA-LSTMs (Choi, 2018; Dave et al., 2021; Fan et al., 2021).

2.2 Machine Learning Models

This subsection refers to Chen and Guestrin, 2016.

Gradient boosting methods are able to model nonlinear relationships without strong parametric assumptions. They offer flexibility but often require extensive feature engineering for optimal performance. For a given data with m predictors, gradient boosting uses K additive functions f (weak learners, also known as decision trees) to estimate the dependent variable

$$\hat{y}_t = \sum_{k=1}^{K} f_k(\mathbf{x}_t), f_k \in \mathcal{F},$$

where space of functions $\mathcal{F} = \{f(\mathbf{x}_t) = w_{q(x_t)} | q : \mathbb{R}^m \to \{1, \dots J\}, w \in \mathbb{R}^J\}$, where

$\mathbf{x}_t \in \mathbb{R}^m$ - Predictor input vector at time t,

$w = w_1, \dots w_J$ - Prediction values (also called weights) assigned to each leaf,

J - Number of leaves in the tree,

q - Independent tree structure that maps inputs to J leaf indices.

For gradient tree boosting, the model is trained in an additive manner. Let $\hat{y}_t^{(k)}$ be the prediction of the t-th instance at the k-th iteration, by adding f_k the following loss equation can be minimized:

$$\mathcal{L}^{(k)} = \sum_{t=1}^{T} l\left(y_t, \hat{y}_t^{(k-1)} + f_k\left(\mathbf{x}_t\right)\right) + \Omega\left(f_k\right)$$

where

$\mathcal{L}^{(k)}$ - Total loss at k-th iteration,

$l(y_t, \hat{y}_t)$ - Individual loss for every time t, where $\hat{y}_t = \hat{y}_t^{(k-1)} + f_k\left(\mathbf{x}_t\right)$;

y_t - Observed value at time t,

$\hat{y}_t^{(k-1)}$ - Estimated value for the t-th time at iteration $k-1$,

$f_k(\mathbf{x}_t)$ - Estimation made by the k-th function for the time t,

$\Omega(f_k)$ - Regularization term (penalizes complexity of f_k),

$\Omega(f_k) = \gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} w_j^2$,

where γ and λ are regularization parameters (for trees and weights).

Second-order approximation can be used to optimize the the equation in general setting.

$$\mathcal{L}^{(k)} \simeq \sum_{t=1}^{T}\left[l\left(y_t, \hat{y}^{(k-1)}\right) + g_t f_k\left(\mathbf{x}_t\right) + \frac{1}{2} h_t f_k^2\left(\mathbf{x}_t\right)\right] + \Omega\left(f_k\right),$$

where

$g_t = \frac{\partial}{\partial \hat{y}_t^{(k-1)}} l\left(y_t, \hat{y}_t^{(k-1)}\right) \quad h_t = \frac{\partial^2}{\left(\partial \hat{y}_t^{(k-1)}\right)^2} l\left(y_t, \hat{y}_t^{(k-1)}\right)$.

We can remove the constants to obtain the following simplified equation at step t:

$$\tilde{\mathcal{L}}^{(k)} = \sum_{t=1}^{T} \left[g_t f_k \left(\mathbf{x}_t \right) + \frac{1}{2} h_t f_k^2 \left(\mathbf{x}_t \right) \right] + \Omega \left(f_k \right). \tag{1}$$

Define $I_j = \{ t \mid q \left(\mathbf{x}_t \right) = j \}$ as the instance set of leaf j. We can rewrite equation (1) by expanding Ω as follows:

$$\begin{aligned} \tilde{\mathcal{L}}^{(k)} &= \sum_{t=1}^{T} \left[g_t f_k \left(\mathbf{x}_t \right) + \frac{1}{2} h_t f_k^2 \left(\mathbf{x}_t \right) \right] + \gamma J + \frac{1}{2} \lambda \sum_{j=1}^{J} w_j^2 \\ &= \sum_{j=1}^{J} \left[\left(\sum_{t \in I_j} g_t \right) w_j + \frac{1}{2} \left(\sum_{t \in I_j} h_t + \lambda \right) w_j^2 \right] + \gamma J. \end{aligned}$$

For a fixed tree structure $q(\mathbf{x})$ (meaning split points and sample leaves are known), we can compute the optimal weight w_j^* of leaf j by

$$w_j^* = - \frac{\sum_{t \in I_j} g_t}{\sum_{t \in I_j} h_t + \lambda},$$

and calculate the corresponding optimal value by

$$\tilde{\mathcal{L}}^{(k)}(q) = - \frac{1}{2} \sum_{j=1}^{J} \frac{\left(\sum_{t \in I_j} g_t \right)^2}{\sum_{t \in I_j} h_t + \lambda} + \gamma J. \tag{2}$$

.

Equation (2) can be used as a scoring function to measure the quality of a tree structure q. Normally it is impossible to enumerate all the possible tree structures q. Due to this, a greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used. Assume that I_L and I_R are the instance sets of left and right nodes after the split. Letting $I = I_L \cup I_R$ we get a loss reduction after the split as following:

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{t\in I_L} g_t\right)^2}{\sum_{t\in I_L} h_t + \lambda} + \frac{\left(\sum_{t\in I_R} g_t\right)^2}{\sum_{t\in I_R} h_t + \lambda} - \frac{\left(\sum_{t\in I} g_t\right)^2}{\sum_{t\in I} h_t + \lambda}\right] - \gamma.$$

In practice this is used for evaluating split candidates.

2.2.1 LightGBM

This subsection refers to Ke et al., 2017.

Light Gradient Boosting Machine (LightGBM) is a gradient boosting decision tree algorithm that addresses the challenges of large-scale data in machine learning tasks. On top of the gradient boosting decision tree it uses two novel techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). In GOSS, training instances are ranked according to the absolute values of their gradients in descending order. Then top round$(a \cdot n), a \in (0,1)$ instances with larger gradients are kept to get an instance subset A. From the remaining set A^c consisting round$((1-a)\cdot n)$ instances with smaller gradients, a subset B with size $b \times |A^c|, b \in (0,1)$ is randomly sampled. Lastly, instances are split according to the estimated variance gain of splitting predictor i at threshold d for a fixed node of the decision tree $\hat{V}_i(d)$ computed over the subset $A \cup B$ as

$$\hat{V}_i(d) = \frac{1}{2}\left[\frac{\left(\sum_{t\in I_L} g_t^*\right)^2}{\sum_{t\in I_L} h_t + \lambda} + \frac{\left(\sum_{t\in I_R} g_t^*\right)^2}{\sum_{t\in I_R} h_t + \lambda} - \frac{\left(\sum_{t\in I} g_t^*\right)^2}{\sum_{t\in I} h_t + \lambda}\right] - \gamma,$$

where

$$g_t^* = \begin{cases} g_t, & \text{if } t \in A, \\ \dfrac{1-a}{b}\, g_t, & \text{if } t \in B, \end{cases}$$

and

I_L, I_R are the instance sets of left and right nodes after the split.

In GOSS, estimated $\hat{V}_i(d)$ is used over a smaller instance subset, instead of the accurate $V_i(d)$ over all the instances to determine the split point. This largely reduces the computation cost.

Due to high-dimensional data being usually very sparse, LightGBM uses a nearly lossless approach to reduce the number of features (exclusive feature building). This is possible, as in a sparse feature space, many features are mutually exclusive, so they never take nonzero values simultaneously. This enables to safely bundle exclusive features into a single feature. LightGBM uses a histogram-based tree construction approach, where continuous features are discretized into bins. The complexity of histogram building changes from $O(Tf)$ to $O(Ts)$, while $s < f$, where f is the number of features and s is the number of bundles. This speeds up training of gradient boosted decision trees significantly without hurting the accuracy.

2.2.2 XGBoost

This subsection refers to Chen and Guestrin, 2016.

Extreme gradient boosting (XGBoost) is another gradient boosting decision tree algorithm which is designed to optimize speed and performance in machine learning tasks such as regression, classification, and ranking. On top of gradient boosted decision trees, XGBoost uses split finding algorithms like basic exact greedy algorithm, approximate algorithm and weighted quantile sketch.

Basic exact greedy algorithm enumerates over all the possible splits on all the features. To enumerate all splits for continuous features, the algorithm

sorts data according to feature values first, to visit the data in sorted order to accumulate the gradient statistics.

Approximate algorithm is involved for memory purposes as for the last algorithm the data might not fit into memory. Approximate algorithm finds candidate splitting points according to percentiles of feature distribution. Then the continuous features are mapped into buckets split by the candidate points. After this, statistics are aggregated and best solution base on that is found. There are two variants of this algorithm: global and local. Global variant proposes all the splits during the initial phase of tree construction and uses same proposals for splits finding at all levels. Opposed to global variant, local variant re-proposes after every split.

Weighted quantile sketch is used to propose M candidate split points indexed by j for approximate algorithm. Commonly, percentiles of a feature are used to make candidates distribute evenly on the data. We can define rank functions $r_i : \mathbb{R} \to [0, +\infty)$ as

$$r_i(z) = \frac{1}{\sum_{t=1}^{T} h_t} \sum_{\substack{t=1 \\ x_{ti}<z}}^{T} h_t,$$

which represent the proportions of instances whose predictor i value for instances $t = 1, \ldots, T$ is smaller than z. The goal is to find candidate split points $\{s_{i1}, s_{i2}, \cdots s_{iM}\}$ so

$$|r_i(s_{i,j}) - r_i(s_{i,j+1})| < \epsilon, \quad s_{i1} = \min_{1 \le t \le T} x_{ti}, \quad s_{iM} = \max_{1 \le t \le T} x_{ti},$$

where ϵ is an approximation factor, meaning that there are about $\frac{1}{\epsilon}$ candidate points. Each data point is weighted by h_t.

Sparsity-aware split finding algorithm is used for many real-world problems, as it is common for input x to be sparse. This might be due to presence of missing values in the data, frequent zero entries in the statistics and artifacts of feature engineering. Due to this, it is important to make the algorithm aware of the sparsity pattern in the data. For this, a default direction is added in each tree node. When a value is missing in the sparse matrix x, the instance is classified into the default direction. There are two choices of default direction in each branch. The optimal default directions are learned from the data. Only the non-missing entries are visited. Non-presence is treated as a missing value. Same algorithm can be applied when the non-presence corresponds to a user specified value by limiting the enumeration to consistent solutions.

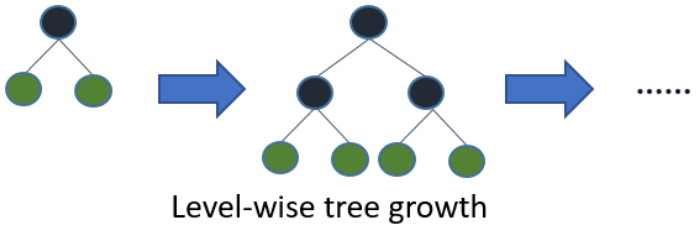
The main difference between LightGBM and XGBoost splitting algorithms can be seen on figure 1.

2.3 Deep learning Model

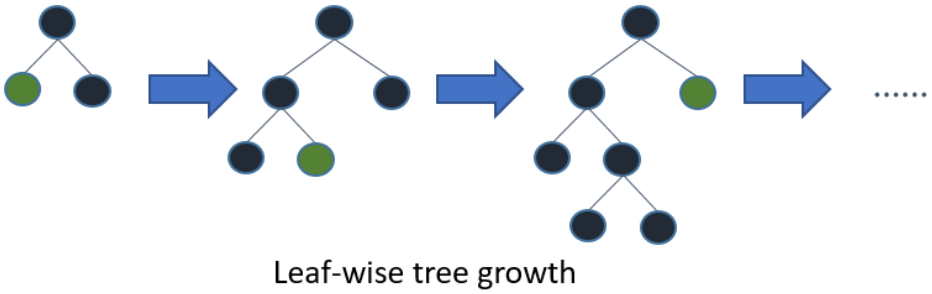
This subsection refers to Haykin, 2009.

Deep learning is used to describe a field of machine learning, which uses artificial neural networks. Neural networks (NNs) are a class of machine learning models inspired by the structure and function of the brain, where interconnected neurons process and pass on information (McCulloch and Pitts, 1943). The most commonly used NN architecture is the feedforward neural network (FNN) (Figure 2).

In this architecture input neurons x_t are connected to one or more layers of hidden neurons h_k, which in turn are connected to output neurons y_j (Figure 3).



Level-wise tree growth in XGBOOST.



Leaf wise tree growth in Light GBM.

Figure 1: XGBoost and LightGBM tree growth architectures. Figure from Khandelwal, 2017

Each neuron k receives input signals from m input variables $x_t, t = 1, \ldots, T$, where each input x_t is connected to neuron k through a weighted link with associated weight w_{kt}. Each hidden neuron applies a transfer function (activation function φ) to transform the inputs it receives. One of the most widely used activation functions is the hyperbolic tangent function

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

which introduces nonlinearity into the model and enables learning of complex patterns.

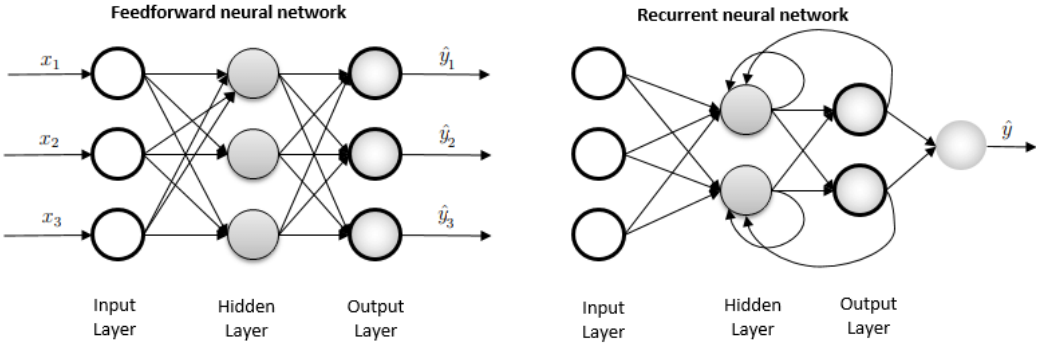


Figure 2: Example of an architecture of a 3-3-3 feedforward and 3-2-2-1 recurrent neural network. Original figure from Pekel and Kara, 2017

These associated weights represent the influence of each input on the neuron's activation. In addition to weighted inputs, bias term $b_k = w_{k0}$ is used to adjust the neuron's total input. The bias can also be expressed as a fixed input $x_0 = +1$. Using this we can construct the formula for the output of neuron k

$$y_j = \varphi(\sum_{t=0}^{T} x_t w_{kt})$$

(Hsieh, 2004).

2.3.1 Finding optimal weights and biases

Goodfellow, Bengio, and Courville, 2016

In FNN, weight matrix W is first initialised, after which neuron k output is calculated. Output from the first pass (forward propagation) is used to get a loss between output $\hat{y}$ and true value y. Using losses of every point in the dataset we can calculate the cost function

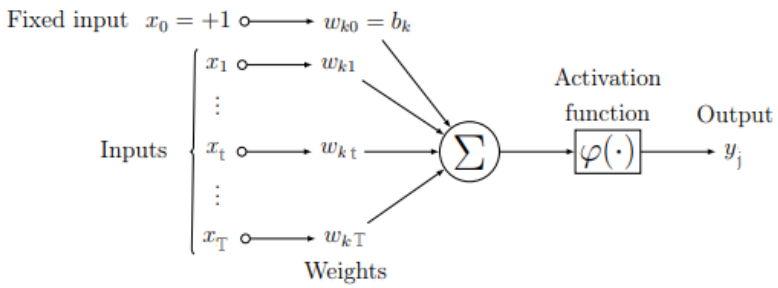


Figure 2. Model of an artificial neuron k.

Figure 3: Architecture of a neuron k. Original figure from Haykin, 2009, figure from Peedosk, 2019

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_t, \hat{y}_t),$$

where θ is the matrix of weights and biases.

Then derivatives of the cost function $\mathcal{L}$ (gradients) are computed w.r.t each parameter using backpropagation algorithm. After that each weight and bias are updated using gradient descent.

2.3.2 Backpropagation algorithm

Backpropagation algorithm calculates the gradients w.r.t variables by working backward through the neural network from output. So for example in the case of a neural network with one hidden layer first the gradient w.r.t output activation (φ_o) is

$$\frac{\partial \mathcal{L}}{\partial \varphi_o}.$$

Then chain rule is applied to get the rest of the gradients, so gradient w.r.t output before activation ($\mathcal{S}_o$) is

$$\frac{\partial \mathcal{L}}{\partial \mathcal{S}_o} = \frac{\partial \mathcal{L}}{\partial \varphi_o} \frac{\partial \varphi_o}{\partial \mathcal{S}_o}.$$

Using that we can get weights and biases of the output (W_o and b_o respectively) as follows

$$\frac{\partial \mathcal{L}}{\partial W_o} = \frac{\partial \mathcal{L}}{\partial \mathcal{S}_o} \frac{\partial \mathcal{S}_o}{\partial W_o}$$

and

$$\frac{\partial \mathcal{L}}{\partial b_o} = \frac{\partial \mathcal{L}}{\partial \mathcal{S}_o}$$

respectively.

Next we can find the gradient w.r.t hidden layer activation (φ_h)

$$\frac{\partial \mathcal{L}}{\partial \varphi_h} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \mathcal{S}_o} \frac{\partial \mathcal{S}_o}{\partial \varphi_{hi}}.$$

Finally we can find gradient w.r.t hidden layer before activation ($\mathcal{S}_h$) and hidden layer weights and biases (W_h and b_h respectively)

$$\frac{\partial \mathcal{L}}{\partial \mathcal{S}_h} = \frac{\partial \mathcal{L}}{\partial \varphi_h} \frac{\partial \varphi_h}{\partial \mathcal{S}_h},$$

$$\frac{\partial \mathcal{L}}{\partial W_h} = \frac{\partial \mathcal{L}}{\partial \mathcal{S}_h} \frac{\partial \mathcal{S}_h}{\partial W_h}$$

and

$$\frac{\partial \mathcal{L}}{\partial b_h} = \frac{\partial \mathcal{L}}{\partial \mathcal{S}_h}$$

respectively.

For every iteration, every weight and bias is updated with gradient descent as follows:

$$W_{new} = W - \eta \frac{\partial \mathcal{L}}{\partial W}$$

and

$$b_{new} = b - \eta \frac{\partial \mathcal{L}}{\partial b},$$

where η is learning rate.

One has to be careful with tuning learning rate as if the learning rate is too large, then every correction will overshoot the minimum. But if the learning rate is too small, then the model will fail to converge meaningfully in the given number of epochs (training cycles).

2.3.3 Recurrent neural network

Modeling time series with a feedforward neural network is problematic, as it doesn't account for sequential dependencies. Recurrent neural network (RNN) addresses this issue by allowing feedback loops in the model (typically from hidden layer back to itself, implying h_t is also the output of the RNN) (Figure 2)

$$h_t = \varphi(Wx_t + Uh_{t-1} + b),$$

where

W - Weight matrix associated with input x_t from from time t,

U - Weight matrix associated with hidden state h_{t-1} from time $t-1$ and

b - Bias term.

This is more appropriate to sequential data, as this allows the model to store memory of past inputs in the hidden layer. However, one of the main issues with RNNs is the vanishing gradient problem - gradients of the loss function become exponentially small with backpropagation. This means that the model doesn't use input from further up the sequence. To also account for longer term dependencies, LSTM architecture is used.

2.3.4 Long short term memory

This subsection refers to Hochreiter, 1997 and Gers, Schmidhuber, and Cummins, 2000.

LSTM is a type of recurrent neural network (RNN) designed to model and predict sequential data that has long-term dependencies. In addition to RNNs, LSTMs use an architecture with recurrent memory cells (Figure 4) to maintain and update information over long sequences. To achieve that, in addition to memory cells, LSTMs use gates (forget, input and output gates), which pass information between the memory cells and LSTM's RNN *net*.

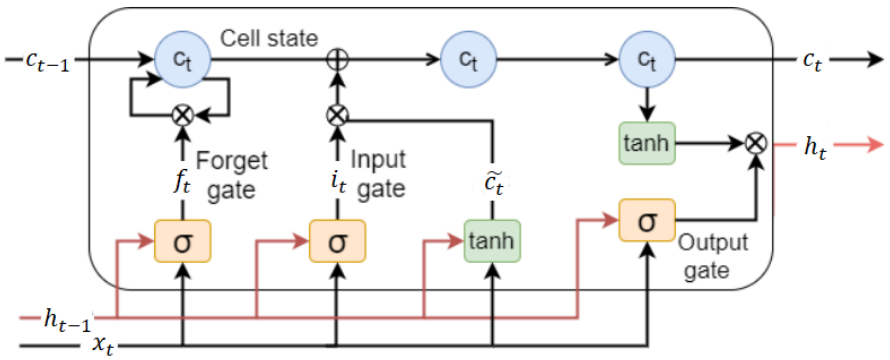


Figure 4: Architecture of a memory cell at time t. Figure is created using base figure from Jenkins et al., 2018

Forget gate unit is used to learn to gradually reset memory blocks once their contents are out of date. For each time t, forget gate f_t is computed as

$$f_t = \sigma\left(net_f(t)\right),$$

where

σ - Sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ and

$$net_f(t) = W_f x_t + U_f h_{t-1} + b_f,$$

where

W_f - Forget gate weight matrix linked to input x_t for time t,

U_f - Forget gate weight matrix linked to hidden state h_{t-1} for time $t-1$,

b_f - Forget gate bias term.

Each memory cell for time t has a recurrently self-connected linear unit called the "Constant Error Carousel" (CEC), whose activation is called the cell state c_t. A multiplicative input gate unit is used to protect memory contents stored in CEC from perturbation by irrelevant inputs. For each time t, input gate i_t is computed as

$$i_t = \sigma\left(net_i(t)\right),$$

where

$$net_i(t) = W_i x_t + U_i h_{t-1} + b_i,$$

W_i - Input gate weight matrix linked to input x_t for time t,

U_i - Input gate weight matrix linked to hidden state h_{t-1} for time $t-1$,

b_i - Input gate bias term.

The aim of the CEC is to preserve gradients over long sequences. For $c_0, t = 0$ and using forget and input gates, the cell state activation $c_t, t > 0$ is given as the following:

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t,$$

where

candidate cell state $\tilde{c}_t = \tanh\left(net_c(t)\right)$ and

$$net_c(t) = W_c x_t + U_c h_{t-1} + b_c,$$

where

tanh - hyperbolic tangent activation function,

W_c - Cell state weight matrix linked to input x_t for time t,

U_c - Cell state weight matrix linked to hidden state h_{t-1} for time $t-1$,

b_c - Candidate cell state bias term.

A multiplicative output gate unit is used that protects other units from perturbation by currently irrelevant memory contents stored in t. For each time t, output gate o_t is computed as

$$o_t = \sigma\left(\text{net}_o(t)\right),$$

where

$$\text{net}_o(t) = W_o x_t + U_o h_{t-1} + b_o,$$

W_o - Output gate weight matrix linked to input x_t for time t,

U_o - Output gate weight matrix linked to hidden state h_{t-1} for time $t-1$,

b_o - Output gate bias term.

Thus we get the hidden state h_t for time t as the final output

$$h_t = o_t \tanh(c_t).$$

In the beginning of learning phase, error reduction may be possible without storing information over time. Thus the network will abuse memory cells as bias cells. It may take time to release abused memory cells to make memory cells available for further learning. Similar issue happens when two adjacent memory cells store the same redundant information. To combat this, it is necessary to initialize the forget, input or the output gate with a negative bias to push initial memory cell activations towards zero.

2.4 Hybrid Model

This subsection refers to Zhang, 2003.

Hybrid models capture linear trends and seasonality by considering a time series to be composed of a linear autocorrelation structure and a nonlinear component.

$$y_t = L_t + N_t + \epsilon_t,$$

where L_t denotes the linear component,

N_t denotes the nonlinear component and

ϵ_t is random error.

These two components have to be estimated from the data. First, SARIMA models the linear component, then the residuals from the linear model will

contain only the nonlinear relationship

$$e_t = y_t - \hat{L}_t,$$

where e_t is the residual at time t from the linear model and

$\hat{L}_t$ is the forecast value for time t from the estimated relationship.

With n input values, the machine and deep learning models for the residuals will be

$$e_t = \hat{N}_t + \epsilon_t = f(e_{t-1}, e_{t-2}, ..., e_{t-n}) + \epsilon_t,$$

where f is a nonlinear function determined by machine or deep learning model and ϵ_t is the random error. The combined forecast is then the following:

$$\tilde{y_t} = \hat{L}_t + \hat{N}_t.$$

The hybrid model exploits the unique feature and strength of ARIMA model as well as non-linear models in determining different patterns. This should improve the forecasting performance compared to only applying either of the methods.

2.5 Loss functions

The most common way to evaluate the performance of the models from different methods in using loss functions. Loss functions are especially important for machine learning methods and deep learning methods as they are also used for training models.

Loss function is defined as $L(y, \hat{y})$, where y is the true value and $\hat{y}$ is predicted value of a model. In it's simplest form loss function is a penalty for prediction error. The aim of the loss function is to show the discrepancy of true values and predicted values. Therefore the lower the value of loss function, the lower the discrepancy. Due to this the goal is to minimize the loss function.

The most suitable loss functions for regression tasks are bilateral loss functions (Nie, Hu, and Li, 2018). The most widely used bilateral loss functions for regression tasks, especially when no set of estimates are known to be the most reliable are root-mean-square error (RMSE) and the mean absolute error (MAE) (Willmott and Matsuura, 2005). RMSE is more sensitive to outliers compared to MAE (Willmott and Matsuura, 2005). This is not ideal for time series tasks in this thesis as there is larger volatility from noise due to the nature of the market data.

MAE involves summing the magnitudes (absolute values) of the errors and then dividing the total error by n (Willmott and Matsuura, 2005)

$$L(y, \hat{y}) = |y - \hat{y}|.$$

Therefore MAE is calculated as following:

$$\text{MAE} = \frac{1}{T} \sum_{t=1}^{T} |y_t - \hat{y}_t|.$$

Due to the robustness of MAE and it being one of the simplest widely used loss functions, this thesis will use it for machine learning and deep learning modeling. As different financial instruments and portfolios have very different scales, a different loss function is needed for evaluating the performance of models. To get fair comparisons, this thesis will use mean absolute scaled

error (MASE), as it is closely related to MAE, while being independent from scale of the variables

$$L(y, \hat{y}) = \frac{|y - \hat{y}|}{\frac{1}{T-1}\sum_{t=2}^{T} |y_t - y_{t-1}|}.$$

Therefore MASE is calculated as following:

$$\text{MASE} = \frac{\frac{1}{T}\sum_{t=1}^{T} |y_t - \hat{y}_t|}{\frac{1}{T-1}\sum_{t=2}^{T} |y_t - y_{t-1}|}.$$

MASE is made to combat the shortcomings of relative measures and measures based on relative errors (Hyndman and Koehler, 2006). It achieves this by scaling the error based on the in-sample MAE from the naive forecast method (Hyndman and Koehler, 2006).

Machine learning and deep learning models struggle to learn with the presence of strong trends in series (Zhou, 2023, Elsworth and Güttel, 2020) and often assume certain degree of stationarity (Liu et al., 2023). For both machine learning and deep learning models it is recommended to remove such trends via differencing (Brown et al., 2024, Elsworth and Güttel, 2020, Schmid et al., 2024). As we expect strong linear trends in daily profit data for financial institutions, we can modify MASE to better fit the purpose of this thesis. This thesis will use rolling forecast strategy to better mimic forecasting in day-to-day operations of a financial institution.

Let $\Delta y_t = y_t - y_{t-1}$, then we can construct the forecasted value $\hat{y}$ of a model to reflect first order of difference instead of using the actual value. Rolling forecast means that the model inputs (or even the model itself) are updated at every step to use all the available data. Due to the nature of rolling forecast, we can assume that true value y_{t-1} is known for time $t-1$ at every step

of the forecast. Therefore we can construct the forecasted difference at every time step $\hat{\Delta y}_t = \hat{y}_t - y_{t-1}$ (not to be confused with $\Delta\hat{y}_t = \hat{y}_t - \hat{y}_{t-1}$). Using this, MASE can be expressed the following way:

$$\text{MASE} = \frac{\frac{1}{T}\sum_{i=1}^{T}|(\Delta y_t + y_{t-1}) - (\hat{\Delta y}_t + y_{t-1})|}{\frac{1}{T-1}\sum_{i=2}^{T}|\Delta y_t|} = \frac{\frac{1}{T}\sum_{t=1}^{T}|\Delta y_t - \hat{\Delta y}_t|}{\frac{1}{T-1}\sum_{t=2}^{T}|\Delta y_t|}.$$

MASE was proposed by Hyndman and Koehler, 2006 with an assumption that the MAE for the first in-sample forecast is not known for the naive method, which is not the case for any series data that naturally starts before the range used in research. We propose a new improved metric that can be used if the value before the series is also known. Proposed corrected MASE (cMASE) can be defined as the following:

$$\text{cMASE} = \frac{\frac{1}{T}\sum_{t=1}^{T}|y_t - \hat{y}_t|}{\frac{1}{T}\sum_{t=1}^{T}|y_t - y_{t-1}|} = \frac{\sum_{t=1}^{T}|y_t - \hat{y}_t|}{\sum_{t=1}^{T}|y_t - y_{t-1}|},$$

which has the added property of being exactly 1 for the naive method (since in that case $\hat{y}_t = y_{t-1}$), therefore being more intuitively interpreted than the original MASE metric. This also means that any values over 1 imply worse performance than the naive method and any values under 1 imply better performance compared to naive method.

Using the newly defined cMASE for our final performance metric in case of differencing can be expressed as the following:

$$\text{cMASE} = \frac{\sum_{t=1}^{T}|\Delta y_t - \hat{\Delta y}_t|}{\sum_{t=1}^{T}|\Delta y_t|}.$$

2.6 Importance and hypothesis

Forecasting accurate profits is important to make more informed investment decisions. For financial institutions working with market data it is also essential in order to get a better understanding of the current market climate. It is also useful to detect potential market data errors or anomalies in a more timely manner.

This thesis attempts to explore how traditional time series methods compare against more modern machine learning and deep learning models in different type of financial series - unlike most studies, which focus on individual categories of models, this thesis will give a more comprehensive overview of using different methods to forecast financial time series data with different levels of aggregation. Even more, this thesis attempts to keep comparisons as fair as possible by implementing as similar method as possible to machine learning, deep learning and hybrid models in addition to using a time sensitive cross-validation for all the different methods.

To achieve this, we will create a flexible modular framework to be used in Swedbank AB for not only forecasting profits, but any series data. Even though there exists automated forecasting pipelines (Meisenbacher et al., 2022), this thesis doesn't attempt to automate the forecasting process as there are too many special cases to consider in terms of market data. As training and testing machine and deep learning models can be very costly, the thesis will also attempt to find out how long of a training and testing series is reasonable for data splitting in time series data and characterize how final testing results change with the size of training data used.

Due to latest research suggesting hybrid ARIMA-LSTM models might be the strongest in forecasting related to profits and financial market data (Choi,

2018; Dave et al., 2021; Fan et al., 2021), we propose a hypothesis that the overall best model will be a hybrid ARIMA-LSTM model.

3 Methodology

3.1 Dataset

The models are trained and tested using Swedbank AB market data from Swedish financial markets area. The whole dataset consists of aggregated daily profit and loss figures from 2017-01-01 to 2025-04-30 (100 months). Dataset has 2094 rows from non-holiday weekdays (mostly five values per week). To compare how the models perform with different type of underlying data, we've decided to test models on three time series with different levels of aggregation. The levels used in this thesis are the following: aggregated daily profits from one type of instrument (comprised of numerous different trades with that type instrument), aggregated daily profits from one portfolio and aggregated daily profits from one trading desk.

Due to bonds being less volatile and more tied to macroeconomic factors, this thesis will attempt to assess profit series aggregated on an instrument level modeling using aggregated profit data from government bonds and bond futures used to hedge the asset risk. As mentioned previously, it is unreasonable to look at profits from government bonds without also looking at profits from the instrument used to hedge them. Due to the stable and often linear nature of these instruments, we expect these instruments to have the least amount of noise in the series, which could give an edge to traditional time series models. To assess the performance with profit series aggregated on a portfolio level, this thesis used aggregated profit data from a portfolio that primarily deals with government bonds, but also has a mix of other instruments mentioned earlier (for example but not limited to FX, FRA and swaps). To assess the different method's performance on profit series aggregated on a trading desk

level, this thesis used profit data from an earlier mentioned desk that contains the portfolio, which is used for profit series aggregated on a portfolio level. Data for bonds and bond futures, for the portfolio and for the trading desk were used separately for modeling. This means that all of the methodology was applied to instruments, portfolio and desk one after another.

3.2 Data preprocessing

As the dataset was missing data from weekdays from holidays and small number of weekdays, where market data was erroneous, these weekdays were linearly interpolated using the two closest weekdays where there was data (one before and one after missing row) as follows:

$$\hat{x_t} = \frac{x_{t-a}b + x_{t+b}a}{a+b},$$

where

$\hat{x_t}$ is the interpolated value from time t, using closest existing values from times $t-a$ and $t+b$.

After interpolation the series had 2173 values.

Following interpolation, the series data was checked for stationarity using Augmented Dickey Fuller (ADF) test. ADF tests checks if the unit root is present in the series using the following formula:

$$\Delta y_t = \mu + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-1},$$

where

μ - Constant drift term,

βt - Time trend,

γ - Unit root coefficent (in case of non-stationary series $\gamma = 0$) and

$\sum_{i=1}^{p} \delta_i \Delta y_{t-1}$ - Lagged difference terms up to lag p (Said and Dickey, 1984).

As all three series were not stationary, the series were differenced at lag 1 and re-checked for stationarity. After differencing all the series once, every one of them became stationary. As all of the models perform better on stationary series (Hyndman and Athanasopoulos, 2018), we will use differenced data for all three series for all models.

After differencing the series had 2172 values. All variables were scaled to have a mean of zero and a standard deviation of one. This helps machine and deep learning models converge faster. In addition this makes the models more stable. Without standardization the models could struggle to learn efficiently, leading to underfitting. Variables were standardized as

$$\tilde{y_t} = \frac{y_t - \bar{y}}{s},$$

where

$\tilde{y_t}$ - Standardized value of variable y at time t,

y_t - Value of variable y at time t,

$\bar{y} = \frac{1}{n} \sum_{t=1}^{n} y_t$ and

$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (y_t - \bar{y})^2}$.

The means and standard deviations were estimated before validating every fold and later before each testing segment. This means that validation and testing data were also standardized using the parameters estimated from training data to avoid any potential data leakages during validation and

testing. Data leakage happens when out-of-sample information is used for modeling.

Data was also augmented with lagged values 1-10 ($lagi_t = x_{t-i}$) and higher lags which showed larger residuals in autocorrelation function (ACF) or partial autocorrelation function (PACF) of residuals. Lags were no higher than the average one quarter (3 months) data ($n_{quarter} = \frac{2171 \cdot 3}{100} \approx 65$, so highest lag was 65. Lag 22 was also added to represent monthly data ($n_{month} = \frac{2171}{100} \approx 22$. Similarly, different rolling statistics were added to the data with same times as lags after 1 (for example $maxi_t = \max(x_{t-1}, ..., x_{t-i}), i > 1$). The statistics applied to lags were maximum, minimum, standard deviation and mean. To characterize the yearly and weekly patterns better, calendar week numbers $(1, \ldots, 53)$ and days of the week $(1, \ldots, 5)$ were augmented. This resulted in at least 58 augmented variables for all series. A set of augmented variables $\mathcal{X}$ in this case could be the following:

$$\mathcal{X} = \{lag_1, lag_2, \ldots, lag_{65}, max_2, \ldots, std_{53}, std_{65}, week, day\}.$$

3.2.1 Data splitting

Data was split based on time to maintain the nature of series in all segments. There is no consensus on appropriate split ratio. As the time series was quite large, we have decided to use a more conservative split ratio. Data was split using 90/10 split ratio, so first 90 months of series was used for training and validation. The other 10 months were used for testing. This resulted in a training and validation set of 1955 and 217 observations respectively. For fair comparison between SARIMA, machine learning, deep learning and hybrid models, the first 65 rows were removed due to missing values for some rolling

statistics. Rest of the training data was split into 5 equal segments based on time for cross-validation. This resulted in 378 observations for every fold. Cross-validation was used to avoid overfitting by putting more emphasis on out-of-sample testing to improve its generalization ability. Each segment was further split using 90/10 ratio for training and validation. This resulted in 340 observations that were used for training and 38 observations that were used for validation for each fold (Figure 5).

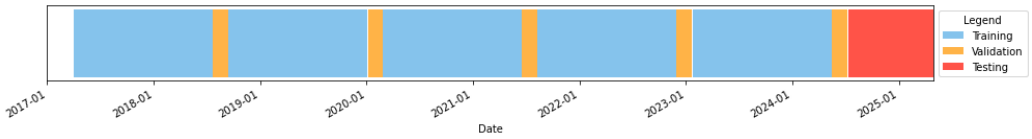


Figure 5: Timewise training, validation, and test splits used for model training and evaluation.

3.2.2 Training data for final testing

Importance of size of the final training data was also tested using the sizes of previously defined folds. For the first testing segment only the chronologically last training fold was used as the training data for final testing (Figure 6).

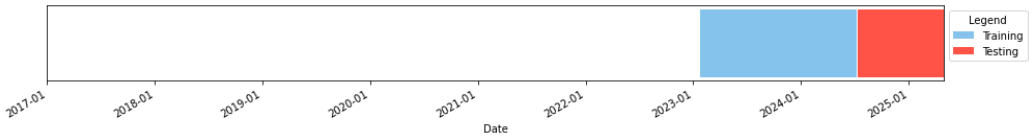


Figure 6: Training data used for testing segment 1.

For the second testing segment, the chronologically last two folds were used as the training data for the same final testing (Figure 7). Similar testing was done with third and fourth testing, with fifth testing using the whole training data.

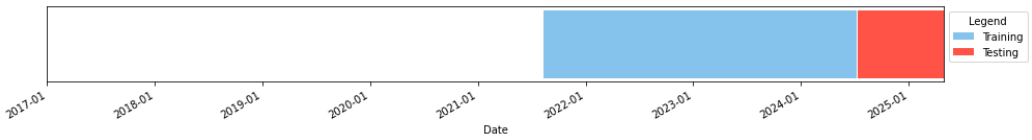


Figure 7: Training data used for testing segment 2.

3.3 Modeling

There are several methods used to improve the machine and deep learning models' performance. We have already introduced augmentation to prevent underfitting. As this resulted in over 58 new variables in dataset that are quite possibly highly correlated, there are several methods used in modeling to prevent overfitting.

3.3.1 Feature importance

Due to the large number of augmented variables used, it is not reasonable to pass all of them to machine learning and deep learning models due to risk of overfitting. To keep the model complexity optimal, a concrete number of variables was used for training and validation, later for testing. To keep the comparison between models as equal as possible, this number of features was determined using permutation importance for all models (Altmann et al., 2010). The benefit of permutation importance is also being able to see which variables affect the forecasts and their impact to the models. After fitting a model to training data, variable values were shuffled across all observations. This ensured that any structure learned from that variable would no longer influence model performance. After each shuffling, model performance was re-evaluated - the larger the decrease in performance, the more important the variable. To avoid instability due to randomness, permutation and evaluation process were repeated several times for each variable, and the average drop

in performance was recorded as the variable's importance. For initial tuning, this was repeated 10 times. For final tuning, this was repeated 30 times. For testing, this was repeated 50 times.

3.3.2 Refitting

Due to the length of the testing set, there is a clear potential issue with model aging. Refitting the models at every time step is very costly for both validation and testing and doing that every single time step increases the risk of overfitting. To counter these issues, we will use refitting time as a time length after which the most important variables are determined and models are refit using the newly made available data from validation and testing set from the time the model was not refit. Inbetween refitting times, the models won't be refit - instead we will use lastly fit model's coefficents/weights/biases on new data from rolling forecast. During refitting hyperparameters such as p or Q for SARIMA or learning rate for non-linear models are not reevaluated. The possible refit times were limited due to the size of validation data from one fold. For example, to avoid potentially refitting at times 12, 24, 36 on validation data with the size of 38, we divided 38 with integers 1-38 to get all the possible refitting time sizes, so a possible refitting time t_i would be the following: $t_i = round(\frac{38}{i}, 1), t = \{1, \ldots, 39\}$.

3.3.3 Regularization

Due to the nature of augmented data, there is a strong possibility that multicollinearity might be an issue when trying to fit machine and deep learning models using over 40 variables. To address this, we will use a combination of Lasso (L1) and Ridge (L2) penalties to shrink coefficients when variables are

correlated. L1 penalty is defined as

$$L1 = L(y, \hat{y}) + \lambda_1 \sum_{i=1}^{n} |\beta_i|$$

(Tibshirani, 1996)

and L2 penalty is defined as

$$L2 = L(y, \hat{y}) + \lambda_2 \sum_{i=1}^{n} \beta_i^2$$

(Hoerl and Kennard, 1970).

A combination of penalties is used as it adds more depth to hyperparameter selection so models could be better at matching data characteristics

$$L1L2 = L(y, \hat{y}) + \lambda_1 \sum_{i=1}^{n} |\beta_i| + \lambda_2 \sum_{i=1}^{n} \beta_i^2.$$

3.3.4 SARIMA

To avoid any potential subjectivity that stems from choosing SARIMA models using ACF and PACF plots, we used cross-validation similarly to other methods for finding the best model. Before cross-validation, every potential model was fit to the whole training data with Ljung-Box test to avoid potential overfitting with unreasonably complex models. For large time series data (over 1000 observations), it is appropriate to use Ljung-Box test with 20 lags (Hassani and Yeganegi, 2020). If any of the first 20 Ljung-Box values were under critical value 0.05, then the potential model was discarded. Models were evaluated on training, validation and testing data using a 1-day rolling origin forecast (Figure 8) - after every forecast, the true value was

added to the training data, which was used to make a forecast until the end of validation and testing segments (Tashman, 2000).

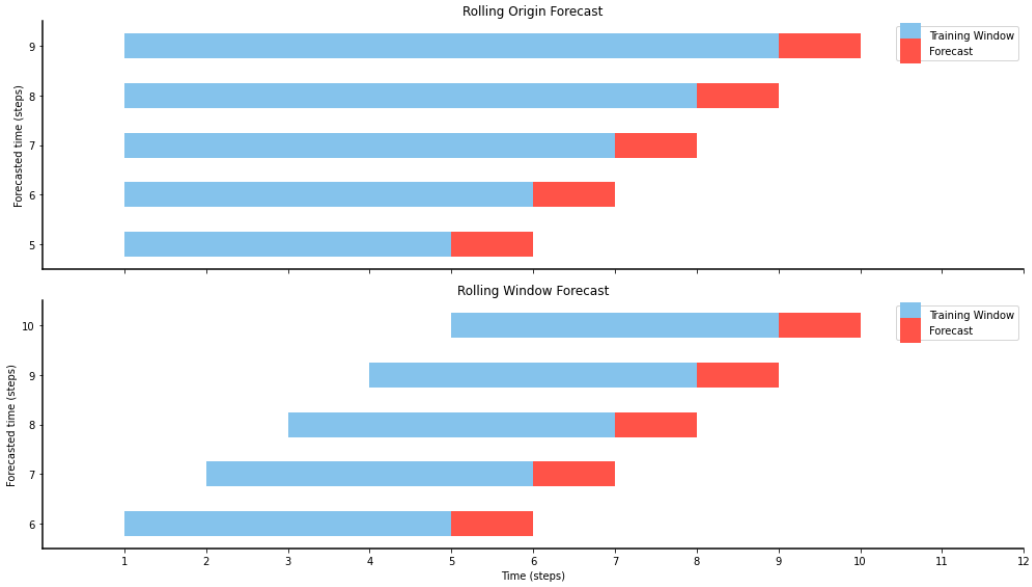


Figure 8: Rolling Origin and Rolling Window Forecast

3.3.5 Machine learning

Machine learning methods used a rolling origin forecast similarly to ARIMA (Figure 8). During validation and testing, before forecasting the next value, all the augmented variables were calculated for time $t + 1$, which were then used to get the forecast for time $t+1$. After forecasting a new day, true value was added as the original series value for $t + 1$.

3.3.6 LSTM

Due to LSTM being trained on a sequence trying to forecast a single value after sequence, it was necessary to divide each training segment into further sequences to avoid overfitting. Due to relatively large folds, to further prevent

overfitting, the sequences were not overlapped. This also avoids any potential data leakage in training. LSTM used tanh and sigmoid activation functions as described previously. LSTM network also used the Adam optimization (Kingma and Ba, 2014) as it has been the prevailing optimizer in neural networks for many years. Adam optimizer computes adaptive learning rates for all variables by keeping track of mean and variance of gradients. Unlike the machine learning models, during validation and testing LSTM used rolling window forecast (Figure 8). For LSTM the sequence up to t with the most important variables was used to forecast $t + 1$ value. After forecasting, true value was added as the original series value for $t+1$ along with the augmented variables for $t+1$ and the first value in sequence was removed for next forecast to keep the sequence data sizes consistent throughout the validation and testing (Tashman, 2000).

In this thesis, LSTM was modeled using Keras framework in Python (Chollet, 2015). Keras makes it is possible to construct neural networks layer by layer. Keras was used for its ease of use and because as it supports LSTM models.

3.3.7 Hybrid models

For Hybrid models, best SARIMA model was fit onto the training data along with 65 observations right before the training data for each fold to get the residuals e_t for training

$$e_t = y_t - \hat{y}_t.$$

After getting the residuals, augmented variables were created and the first 65 observations were removed. SARIMA model was refit using the refitting time and hyperparameters from previously found best SARIMA model. For

each forecast, first SARIMA was used to forecast the value at time $t+1$. Then a previously described rolling forecast was used for machine learning (origin) or deep learning (window) models to forecast residual at time $t+1$. Final forecast of $t+1$ was the sum of SARIMA forecast and residual forecast for $t+1$.

3.4 Hyperparameter tuning

There is no clear consensus on the best way to find the most optimal hyperparameters. The classical way would be using grid search for tuning purposes, however this is computationally very expensive with more variables. As customization was a requirement for this thesis, this thesis used Optuna framework (Akiba et al., 2019) as a more sample efficient method than random search. Differently from random search, Optuna framework builds a probabilistic model to choose hyperparameters better based on previous iterations. Optuna formulates the hyperparameter optimization as a process of minimizing an objective function that takes a set of hyperparameters as an input and returns its performance metric score:

$$x_{opt} \in argmin f(x),$$

where x is a hyperparameter configuration.

Due to the large initial search space and the high-dimensionality of the data, this thesis uses Tree-structured Parzen Estimator (TPE) as Optuna's algorithm to find what hyperparameters to use next using the results from previous trials.

TPE doesn't model $f(x)$ directly, but tries to model two probability densities:

$$\text{bad}(x) = p(x|y \geq y^{\gamma})$$

and

$$\text{good}(x) = p(x|y < y^{\gamma}),$$

where y is observed performance metric score, y^{γ} is a threshold that separates good and bad trials (Watanabe, 2023).

Probability densities are fitted using kernel density estimaton (KDE) as follows:

$$\hat{f} = \frac{1}{nh} \sum_{i=1}^{n} K \frac{x - x_i}{h},$$

where K is the Kernel and $h > 0$ is a smoothing parameter (Hastie, Tibshirani, and Friedman, 2001).

This thesis used the default KDE kernel in TPE - the Gaussian kernel defined as $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$.

After fitting KDEs to estimate $\text{bad}(x)$ and $\text{good}(x)$, the algorithm tries to maximize the ratio $\frac{good(x)}{bad(x)}$ (Watanabe, 2023).

The main benefits of Optuna for this task were compatability with modular programming due to the unique method deployed in this thesis and the option to stop training iteration after first fold or two, if the performance metric gave undesirable results.

Models with given hyperparameters were fit and validated for each fold separately and evaluated according to the average cMASE score in cross-validation. Due to very large initial search space, we used 100 Optuna trials for initial tuning. From initial tuning we chose 3 best sets of hyperparameters to set the intervals for hyperparameters for final tuning. For example if

number of features for the 3 best models were 6,8,9 then an interval of 6-9 would be used for final tuning. For final tuning we used 50 trials. The hyperparameters with best performing models on average across all folds were used for final testing.

3.4.1 Hyperparameters tuned

For SARIMA, hyperparameters that were tuned were the following: order of AR term, order of MA term, order of seasonal AR term, order of seasonal differencing and order of seasonal MA term. Season period used was 5 due to it being one week long. Orders of AR term and MA term were tuned in the range 0-4, orders of seasonal AR and MA term were tuned in the range 0-4, order of seasonal differencing was tuned in the range 0-1 and refitting time.

For machine learning methods, parameters that were tuned were the following: number of estimators (decision trees), max depth (maximum number of layers on a tree), learning rate, number of features, regularization parameters α (L1) and λ (L2) and refitting time.

For LSTM, parameters that were tuned were the following: units (hidden states), dropout (randomly excluding neurons for generalization), recurrent dropout (dropout used between LSTM time steps), epochs (how many times weights and biases are updated), learning rate, number of features, number of sequences, L1 and L2 rates and refitting time. Initially complexity of LSTM network allowing several hidden layers was also tuned. Since every network with more than one hidden layer performed consistently significantly worse than a network with only one hidden layer, this was omitted from final list of tunable hyperparameters.

4 Results

The performance of traditional time series, machine learning, deep learning and hybrid models on profit data aggregated on different levels is presented in the following chapter.

4.1 Profit series aggregated on an instrument level

For profit series aggregated on an instrument level, all of the models managed to beat the naive method ($cMASE < 1$) on average across all the cross validation training folds (Table 1).

Table 1: Performance metrics (cMASE) of different methods on profit series aggregated on an instrument level

Simple series	**SARIMA**	**LGB**	**XGB**	**LSTM**	**H-LGB**	**H-XGB**	**H-LSTM**
Features	5	11	11	13	20	13	14
Refitting time	1	10	13	19	10	13	19
CV average	0.9829	0.9699	0.9920	0.9458	0.9728	0.9769	0.9621
Testing segment 1	1.0161	0.9989	**0.9925**	1.0164	1.0382	1.0141	**1.0491**
Testing segment 2	1.0069	**0.9987**	0.9966	1.0547	1.0225	1.0072	1.2285
Testing segment 3	1.0008	1.0058	1.0042	1.0564	**1.0029**	1.0020	1.1831
Testing segment 4	**0.9981**	1.0032	1.0052	**1.0141**	1.0153	0.9989	1.2243
Testing segment 5	0.9982	0.9988	0.9996	1.0150	1.0106	**0.9979**	1.2301

Note. Bold values show best performance for a method among all five testing segments. "H-" indicates hybrid models.

SARIMA used the least amount of features for the final model. The best SARIMA model in validation was $(3,0,0)x(0,1,1)_5$. The effect of parameters D and Q to the final model was almost non-existent, therefore the best model is almost equivalent to $(3,0,0)x(0,0,0)_5$, which means that seasonality played a minuscule role in the final model. SARIMA was the only method that didn't suffer from overfitting when refitting model after every

forecast, therefore the best model was refitted to all of the data at every timestep during testing of SARIMA model and during training, validation and testing of hybrid models. Hybrid models were the most complex (13-20 augmented features), which might suggest that machine and deep learning models failed to detect meaningful patterns in the SARIMA residuals. For non-hybrid machine and deep learning models LSTM had 2 augmented features more for best model in validation, which might point to more risk of overfitting similarly to hybrid models. One of the most surprising results was that machine and deep learning models that were refit every 2-4 weeks performed better than models that were refit more frequently suggesting that underlying patterns in instrument level aggregated profit time series did not change very frequently. This was also observed during testing when 6-8 of the most important features remained same throughout most of the testing for both regular machine learning, deep learning and hybrid models. Another interesting thing to note is that the best refitting time for all hybrid machine and deep learning models was exactly the same as for the regular models. This suggests that the SARIMA residuals might have same underlying patterns as the instrument level aggregated profit series. Thus it is possible that non-hybrid machine and deep learning models might have modeled different underlying patterns compared to SARIMA, that remained within SARIMA residuals. The best result in validation was achieved by regular LSTM models, followed by hybrid LSTM and regular LightGBM models.

From testing results it seemed correct to assume that models with lot of augmented features were prone to overfitting. The only models that managed to beat naive model with any testing segment were SARIMA, LightGBM, XGBoost and XGBoost hybrid model. While XGBoost seems very consistent replicating similar results in testing as it did in validation, from figure 9 we

can see that cMASE scores for XGBoost were almost 1 during the whole testing time, which means that it underfit the series data making forecasts similarly to the naive method.

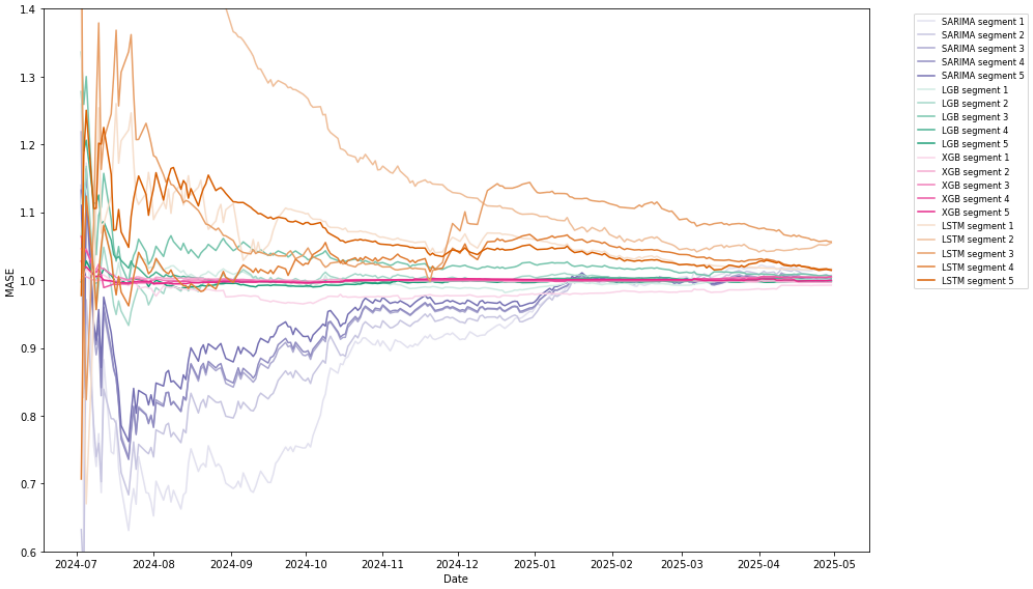


Figure 9: Performance metrics (cMASE) of non-hybrid methods on profit series aggregated on an instrument level over the course of testing

Even though XGBoost hybrid model managed to beat the naive method, from figure 10 we can see that it did so because it was underfitting the SARIMA residuals as well (the H-XGB graph looks almost identical to the earlier observed SARIMA cMASE graph).

During the first three months of testing hybrid LightGBM and hybrid XGBoost models managed to beat the naive model while SARIMA managed to do that for the first six months. SARIMA was even 10% better than the naive model for the first two months (SARIMA, hybrid LightGBM and hybrid XGBoost with testing segment 1 even about 25% better). This suggests that these models should be retrained after every two to three months with series data of a similar size. For the whole testing period the cMASE scores

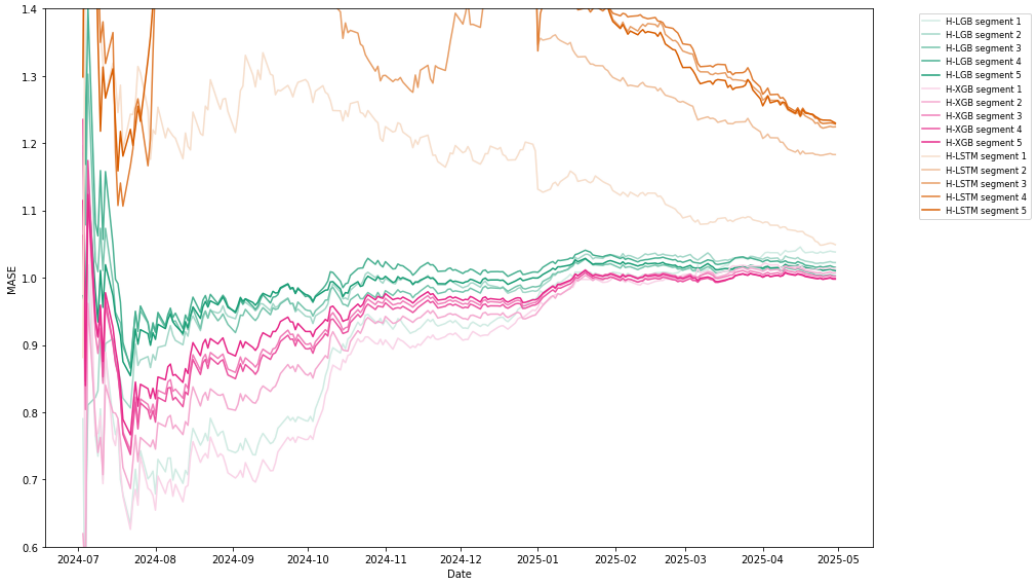


Figure 10: Performance metrics (cMASE) of hybrid methods on profit series aggregated on an instrument level over the course of testing

eventually converged to cMASE similar to what was observed in validation.

Both LSTM and hybrid LSTM models had noticeable negative slope in the second half of testing, which might suggest that with a longer testing time-frame their testing scores might've converged to the cMASE scores that were seen during validation.

Using the average score of testing segment cMASE scores, the best training data to use for testing was the last observed fold in training with an average of 1.0178, however this is largely impacted by the irregular result of hybrid LSTM model. By leaving the performance metrics of hybrid LSTM out, the best testing segment was the testing segment that used all of training data to train the final models with an average score of 1.003, which seems to be consistent with the sentiment that more data for training is better. This effect is most clearly seen with traditional models - for non-hybrid machine and deep learning using only the last training fold had almost exactly the

same performance as using all of the data. While this is surprising that the final models can be tested using only fraction of the data, it could due to recency of the series being important for modeling.

If we leave out the results from XGBoost and hybrid XGBoost due to issues with underfitting, the overall best model for simple instrument level data seems to be SARIMA with the best observed cMASE scores in testing and the most consistent testing result compared to the average cMASE observed in validation followed by regular LightGBM model.

4.2 Profit series aggregated on a portfolio level

For profit series aggregated on a portfolio level, all models except LSTM and hybrid LSTM managed to beat the naive baseline on average across the cross-validation folds (Table 2).

Table 2: Performance metrics (cMASE) of different methods on profit series aggregated on a portfolio level

Inter series	**SARIMA**	**LGB**	**XGB**	**LSTM**	**H-LGB**	**H-XGB**	**H-LSTM**
Parameters	5	6	2	7	6	6	18
Refitting time	1	13	8	19	8	5	10
CV average	0.9973	0.9983	0.9849	1.0085	0.9893	0.9671	1.1651
Testing segment 1	**0.9819**	1.0025	**0.9967**	1.1433	**0.9922**	**0.9846**	1.6037
Testing segment 2	0.9966	1.0185	1.0001	1.0730	0.9855	1.0106	1.0943
Testing segment 3	0.9953	0.9973	0.9999	1.0151	0.9927	1.0024	1.2026
Testing segment 4	0.9967	**0.9916**	0.9999	1.1106	0.9844	0.9960	**1.0615**
Testing segment 5	0.9971	1.0304	1.0002	**1.0133**	1.0001	0.9969	1.1082

Note. Bold values show best performance for a method among all five testing segments. "H-" indicates hybrid models.

Similarly to simple complexity data modeling, SARIMA required 5 parameters and was refit after every forecast. SARIMA was again the only method that didn't suffer from overfitting when refitting model after every forecast,

therefore the best model was refitted to all of the data at every timestep during testing of SARIMA model and during training, validation and testing of hybrid models. The best SARIMA model in validation was $(0,0,3)x(0,0,2)_5$.

It is possible that both LSTM and hybrid LSTM were overfit similarly to simple data as they again had most number of parameters (7 and 18 respectively). The fact that both LSTM and hybrid LSTM cMASE dropped with a larger training set might indicate that there were not enough sequences to meaningfully train the model on the series, which suggests that LSTM would have performed better if there was more series data available. From figure 11 we can see that LSTM actually performed the best between testing months 2-3 (sharp drop in all testing segments), though due to the volatility in LSTM performance metric, it seems unreasonable to make any concrete conclusions from this.

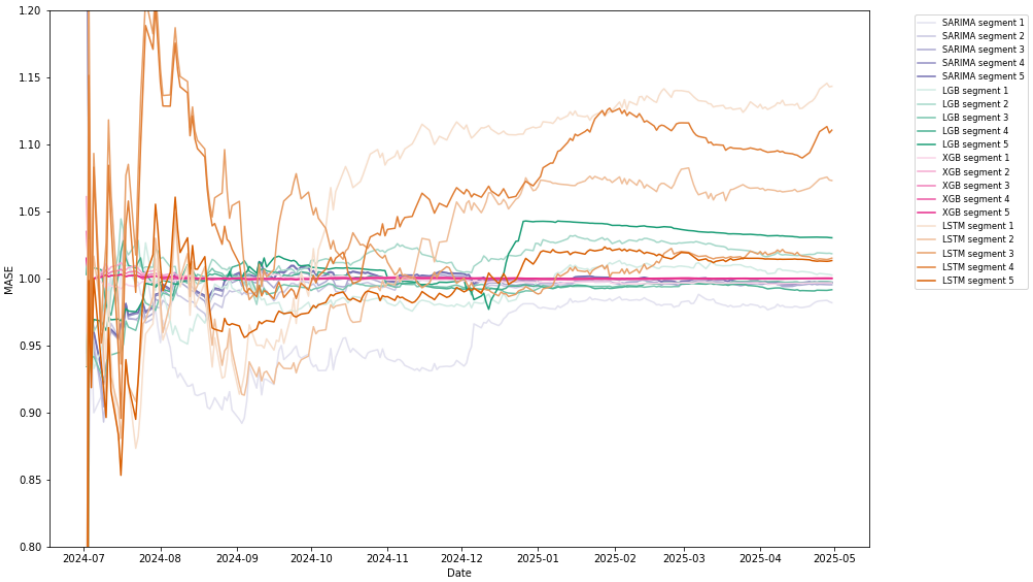


Figure 11: Performance metrics (cMASE) of non-hybrid methods on profit series aggregated on a portfolio level over the course of testing

For profit series aggregated on a portfolio level it seems that while both hybrid

machine learning models were better in validation and testing than their non-hybrid counterparts, they still failed to outperform traditional SARIMA model on the best result in testing. For hybrid machine learning models in testing segment 1, we can see that the cMASE over the course of testing was almost exactly the same as SARIMA, which means the hybrid models failed to find any non-linear patterns from residuals. However, using 4-5 training folds for fitting the initial model for testing resulted in better performance metrics for hybrid machine learning models compared to traditional SARIMA model. This is possibly due to SARIMA model parameters being outdated in later timeframes of testing. All the hybrid models had lower refitting time compared to non-hybrid models. This indicates that there were more pattern changes in residuals compared to original series. Suprisingly similarly to profit series aggregated on an instrument level, on most cases the models were refitted less frequently than once a week, suggesting that on portfolio level the underlying patterns in time series did not change frequently, though it did change more frequently than the series on an instrument level.

From testing results we can see less problems with overfitting compared to profit series aggregated on an instrument level. This is also reflected in the number of parameters - excluding hybrid LSTM, the most number of augmented variables was 7. For portfolio level series all the models except LSTM and hybrid LSTM managed to beat the naive method at some testing segment, with only SARIMA managing to beat naive model in every testing segment. While non-hybrid XGBoost seems to have relatively consistent performance across all testing segments, it might again be due to underfitting as all testing segments are almost one and from figure 11 we can see for these segments the naive model was fitted throughout the whole testing. This underfitting might stem from only 2 parameters used in final model, which

fail to capture any meaningful patterns. Similarly to instrument level series, hybrid XGBoost also failed to find meaningful pattern in the residuals of SARIMA model, as all the cMASE curves of hybrid XGBoost look almost the same as SARIMA cMASE curves (Figure 12).

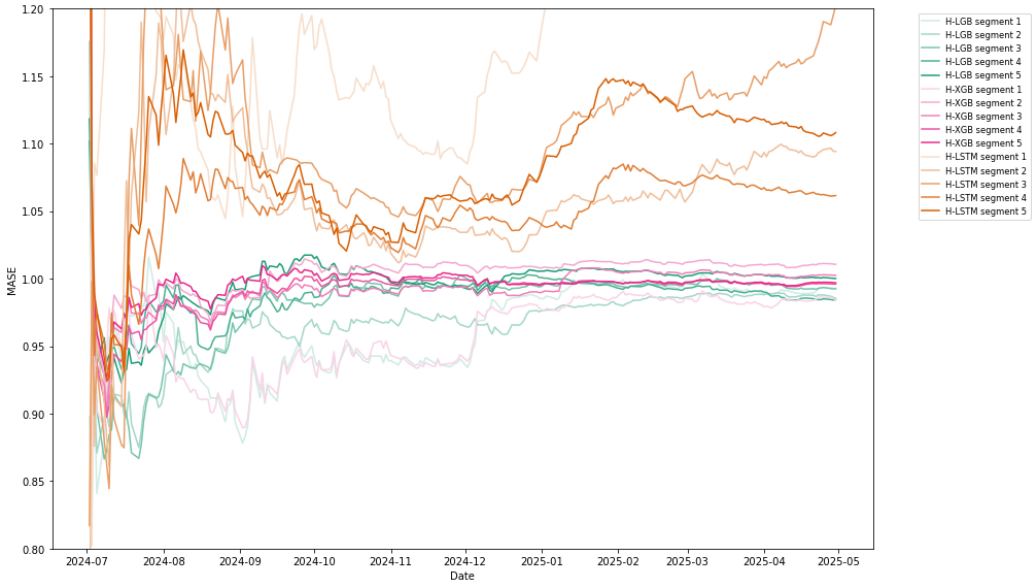


Figure 12: Performance metrics (cMASE) of hybrid methods on profit series aggregated on a portfolio level over the course of testing

Interestingly, hybrid LightGBM performed best on testing segments 2-4, which might suggest that LightGBM might be best for modeling non-linear patterns with anomalous training data.

Similarly to profit series aggregated on an instrument level, excluding deep learning models, we can see SARIMA outperform the naive model over 5% for five months, which might suggest that the initial models can only detect patterns for 5 months (about 110 observations), after which the hyperparameters need to be retuned.

Using the average score of testing segment cMASE scores, the best training data to use for testing was all 5 folds with an average of 1.0201, however this

is again largely impacted by the irregular results of deep learning models. By leaving LSTM and hybrid LSTM performance metrics out, the best testing segment was the testing segment that used only the data from the last fold to train the final models with an average score of 0.9916, which seems to indicate that models can be tested using only the fraction of the training data. This again suggests that recency in the series is important for training.

Since all the models failed to capture meaningful patterns past 5 months of testing data, based on cMASE curves I would say the best model for portfolio level data was hybrid LightGBM as it consistently performed the best for that timeframe in all testing segments.

4.3 Profit series aggregated on a trading desk level

For profit series aggregated on a trading desk level, all models except hybrid LSTM managed to beat the naive baseline on average across the cross-validation folds (Table 3).

Table 3: Performance metrics (cMASE) of different methods on profit series aggregated on a trading desk level

Complex series	**SARIMA**	**LGB**	**XGB**	**LSTM**	**H-LGB**	**H-XGB**	**H-LSTM**
Parameters	4	7	6	12	1	2	17
Refitting time	1	19	19	13	10	8	13
CV average	0.9780	0.9582	0.9924	0.9974	0.9764	0.9724	1.0156
Testing segment 1	0.9780	0.9957	0.9916	1.1195	0.9800	0.9602	1.0663
Testing segment 2	0.9857	0.9859	**0.9902**	1.1236	0.9871	0.9634	1.1392
Testing segment 3	**0.9759**	1.0085	0.9971	1.0219	**0.9750**	**0.9600**	1.1093
Testing segment 4	0.9762	**0.9812**	0.9995	**1.0188**	0.9756	0.9601	1.0093
Testing segment 5	0.9769	1.0057	1.0007	1.0194	0.9806	0.9623	**1.0060**

Note. Bold values show best performance for a method among all five testing segments. "H-" indicates hybrid models.

SARIMA required 4 parameters and similarly to profit series aggregated

on a portfolio level, SARIMA was the only method that did not suffer from overfitting when refitting model after every forecast, therefore the best model was refitted to all of the data at every timestep during testing of SARIMA model and during training, validation and testing of hybrid models. The best model in validation was $(1,0,0)x(1,1,1)_5$. The effect of parameters D and Q to the final model was almost non-existent, therefore the best model is almost equivalent to $(1,0,0)x(1,0,0)_5$.

For SARIMA, XGBoost and all hybrid models we can see that the models performed better at one point in testing compared to validation, which suggests that testing series was more stable due to having less noise. Similarly to portfolio level series all the models except for LSTM and hybrid LSTM were able to beat naive method at least in one testing segment, while only SARIMA and hybrid machine learning models managed to consistently beat the naive method.

Unlike profit series aggregated on an instrument and trading desk level, SARIMA and hybrid XGBoost model managed to consistently beat naive methods by more than 5% for first 5 months of testing, meaning the hyperparameters would need to be retuned less frequently (Figures 11 and 12).

While LSTM and hybrid LSTM models failed to beat the naive method, it seems that both methods got better with more available data suggesting that LSTM might perform much better on a series with more observations. Even though the models had numerous parameters (12 and 17 respectively), they did not overfit the testing series as much as with instrument and portfolio level series.

Hybrid machine learning models had less parameters than their non-hybrid counterparts, which might suggest that they (especially hybrid XGBoost

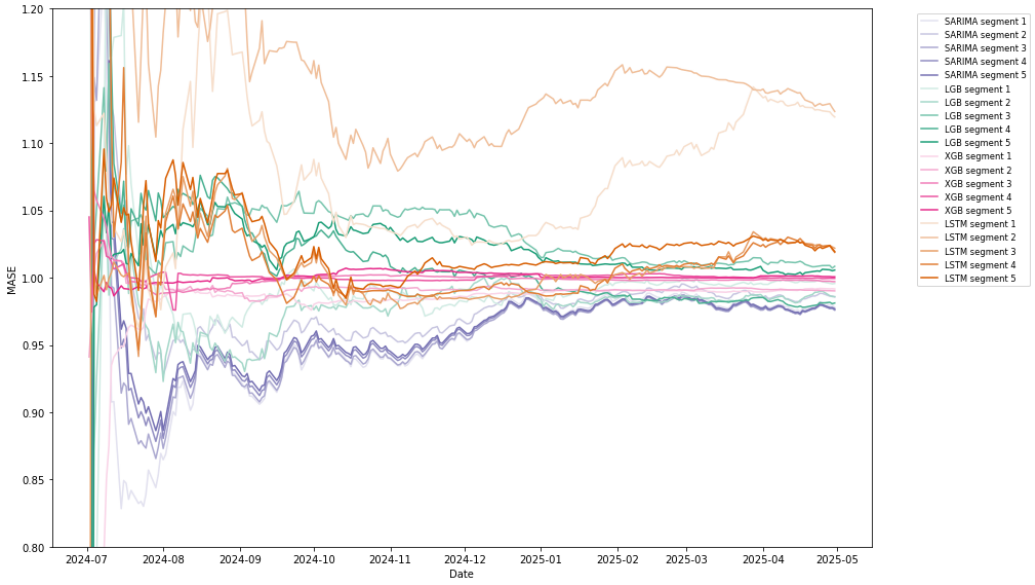


Figure 13: Performance metrics (cMASE) of non-hybrid methods on profit series aggregated on a trading desk level over the course of testing

model) complemented the result from SARIMA quite well. This was also observed with important predictors during testing - for machine learning models, the augmented variables used were mostly either 30+ lag, mean or standard deviation. This suggests that SARIMA modeled series on recent history while machine learning models added context and patterns from longer period. This resulted in hybrid LightGBM beating traditional SARIMA model on some testing segements and hybrid XGBoost beating every other model on every testing segment.

Similarly to profit series aggregated on a portfolio level, all hybrid models had a shorter refitting time compared to non-hybrid counterparts, meaning underlying patterns in residuals might have changed more frequently compared to underlying patterns in series. Refitting time on most models behaves the same way as with previous series - it is most reasonable to refit the machine and deep learning models once every two or more weeks.

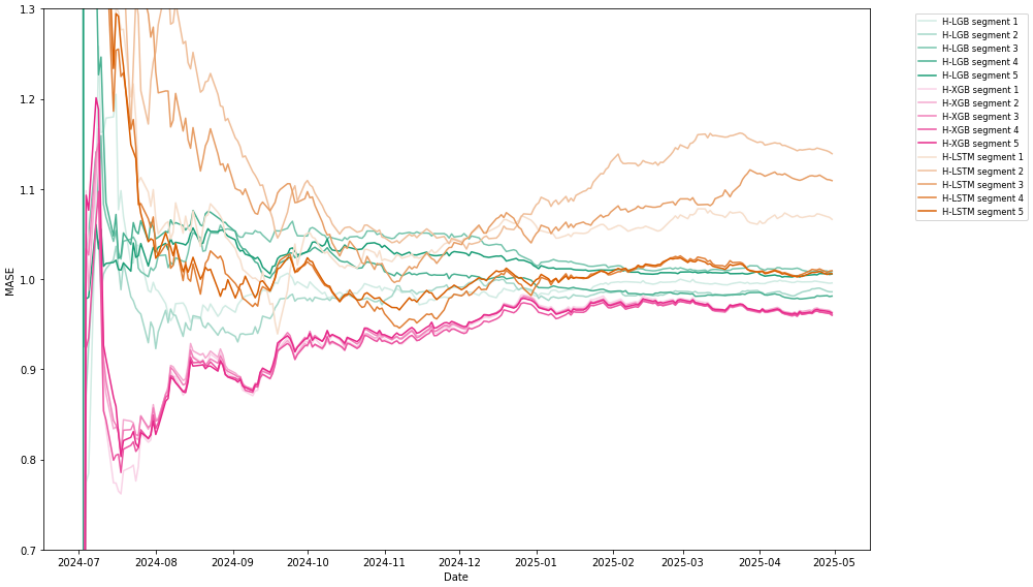


Figure 14: Performance metrics (cMASE) of hybrid methods on profit series aggregated on a trading desk level over the course of testing

Using the average score of testing segment cMASE scores, the best training data to use for testing was all 5 folds with an average of 0.9931, if we leave out the more inconsistent LSTM and hybrid LSTM models, the best testing segment was the testing segment that used only the data from the last 4 folds to train the final models with an average score of 0.9785. While using only 1 fold for training did not result in best score, it had an average score of 0.9811, which is quite close to the scores from testing segment 4 and 5. This means that it might be much more beneficial to use only last fold for final testing purposes due to the much lower testing cost in resources.

Non-hybrid XGBoost model had issues with underfitting similarly to other series, as the cMASE score was almost 1 throughout the testing time. The best model is hybrid XGBoost with the best performance throughout all timepoints in all testing segments.

5 Discussion

Even though series was meant to be scaled for XGBoost models, in the end we chose to forego that as the models were slow to converge and had much worse performance metrics compared to unscaled series model counterparts (on most occasions consistently at least a 30% higher cMASE score). While there have been differences in models due to different scaling methods (Ahsan et al., 2021; Amorim, Cavalcanti, and Cruz, 2023), they have usually found that gradient boosting methods aren't usually affected by different scaling methods. However, there is evidence (Sujon et al., 2024) with small datasets (under 15000), that scaling before using XGBoost to model may result in worse accuracy. The precise reason why scaling consistently resulted in this large of a drop in performance for XGBoost when modeling series data remains unclear. The most likely explanation is that modeling with unscaled data resulted in forecasts similar to naive method, while the scaled version failed to converge to any meaningful results.

For both LightGBM and LSTM scaling the training data resulted in faster convergence and better performance metric results.

While the range of methods used in this thesis may seem a bit extensive, each technique contributed to the improved the models' performance in training and validation while keeping the machine and deep learning models comparable.

In results it was assumed that more parameters might indicate overfitting, which is something that is not inherently true as was observed with LSTM hybrid models profit series aggregated on a trading desk level. This was mostly true for other results as all the extra variables were augmented from a single variable, which suggests large multicollinearity between variables.

This was also mostly true due to the relatively small size (compared to usual machine or deep learning datasets) of the time series.

It was surprising that machine learning, deep learning and hybrid models needed refitting once every two weeks in profit series aggregated on all different levels, though this could be due to the nature of the data as banks do not usually take large risks trading in financial markets.

Profit series aggregated on a trading desk level had the strongest trend for the whole series, while profit series aggregated on an instrument level had the most noise out of all the series. It seems logical that the most stable series had the best performance for most of the models. It also seems logical that models had great difficulties beating naive method on series with the most anomalous data.

Most of the noise for all series happened in periods with high volatility in all of the profit series. This period was between 2021 and 2023, which can be attributed to unprecedented global events such as COVID-19 pandemic and Russo-Ukranian war. In my thesis this falls under training folds 3 and 4. This also means that mainly the testing segments 2 and 3 but also testing segment 4 are most affected by this as these segments use primarily the noisy data for training. This explains why using either only the last fold of data or the whole training data resulted in the best performance metrics. This also indicates that methods, other than LSTM and hybrid LSTM, deployed in this thesis are very reliable in regular time series forecasting as most of them managed to produce meaningful results despite large portion of the training data being anomalous.

In stark contrast to the hypothesis, LSTM and hybrid LSTM models performed the weakest. It is likely this is due to sensitivity of the LSTM method.

Kandadi and Shankarlingam, 2025 highlight that LSTMs tend to overfit on scarce training data, which is why in this thesis we could not replicate the superior performance that Choi, 2018 and Dave et al., 2021 found. While Dave et al., 2021 seemed to have this superior performance due to potential data leakage, Choi, 2018 only forecasted one value ahead using over 50 000 sequences compared to only the tens of sequences used for several consecutive forecasts in this thesis, which might explain why LSTM models in this thesis were more sensitive. While the training set was similarly sized as in research from Fan et al., 2021, the time series of oil and gas well production is less noisy compared to financial data, moreover when that financial series has large portion of data during a pandemic and war. In addition, Fan et al., 2021 did not use rolling forecast for their ARIMA forecast. It could be interesting to see whether the LSTM and hybrid LSTM performance improves if there are more observations from intraday timestamps. It could also be interesting to compare machine learning, deep learning and hybrid models that are modeled using in addition to augmented variables non-augmented variables from market data.

While hybrid XGBoost had excellent performance on profit series aggregated on a trading desk level, I would say that overall, XGBoost and its hybrid models were the worst models in the thesis as they were virtually meaningless - excluding the one hybrid model, XGBoost models forecasted exactly the same as naive method did. For this reason I would recommend using LightGBM instead of XGBoost for any future rolling forecasting tasks featuring more complex relations in time series.

Overall the best method was the traditional SARIMA, due to best overall consistency between validation and testing results, having overall best cMASE scores in the first months of the testing period and in the total test-

ing period. This could be improved in the future by implementing anomaly detection, which would capture shifts in temporal patterns of the time series. Knowing those shifts, it is possible to know the exact times where models should be retrained for best forecasting performance over a long period of time.

6 Conclusions

In this thesis, we have compared the forecasting performance of traditional time series, machine learning, deep learning and hybrid models on daily banking profit data in financial markets area aggregated on three different levels - instrument level, portfolio level and trading desk level.

To evaluate different methods, this thesis uses a novel performance metric corrected mean average scaled error (cMASE), which differs from the regular MASE used with time series by using T one-step naive forecasts instead of $T-1$ forecasts proposed by Hyndman and Koehler, 2006. This gives cMASE better interpretability due to naive method always having performance of $cMASE = 1$, which implies that any model with a score under 1 outperforms the naive method and any model with a score over 1 underperforms the naive method. This is possible as most time series used in modeling have response variable values prior to series used in modeling.

Despite advancements in computational power enabling the use of machine and deep learning models, traditional time series method SARIMA still outperformed the machine learning, deep learning and hybrid methods. SARIMA showed most consistent results between average cross-validation cMASE and testing cMASE. SARIMA is also prefered due to it's better interpretability and smaller computing cost.

Contrary to the initial hypothesis, hybrid ARIMA-LSTM models did not outperform other methods. This was likely due to the sensitivity of LSTM models to relatively scarce and noisy data. The main limitation of this thesis is large portion of the financial data used in training being littered with noise that stems most likely stemming from COVID-19 pandemic and Russo-Ukranian war. This meant that the models performed best on testing set by

either using only training data after the impact of these events or by also using enough training data from before these events to drown out the noise.

Another limitation for this thesis is the long testing period, which resulted in fairly average final performance metrics. Due to the nature of shifting temporal patterns in time series with financial data, the best performing models only performed exceptionally well during first 2-4 months of testing.

The most effective hybrid models were SARIMA - gradient boosting models, where gradient boosting methods complemented SARIMA. This means that SARIMA used recent history for forecasting, which was complemented by gradient boosting methods correcting this forecast by providing context using large lags, rolling means and standard deviations.

While SARIMA models needed refitting after every forecast, machine learning, deep learning and non-linear part of hybrid models performed better when they were refit on average only once every two weeks, which also reduced overall computing cost significantly.

This thesis also confirmed the importance of scaling, which had mixed results with different models. For LightGBM and LSTM scaling improved convergence and performance, while for XGBoost it impaired both convergence and performance by a sizeable margin. Overall XGBoost performed poorly with time series data, which is why the author suggests using LightGBM instead as the gradient boosting method for time series forecasting.

This thesis achieved the task of creating a flexible modular framework for any future time series forecasting. This framework will be improved in the future by adding anomaly detection to characterize the shifts in temporal patterns and adding an option to use non-augmented variables for machine learning, deep learning and hybrid methods.

Citations

Ahsan, Md Mahmudul, Md Parvez Mahmud, Prabir Kumar Saha, Kiran Dip Gupta, and Ziaur Rahman Siddique (2021). "Effect of data scaling methods on machine learning algorithms and model performance". In: *Technologies* 9.3, p. 52.

Akiba, Takuya, Shotaro Sano, Takeru Yanase, Tetsuya Ohta, and Masanori Koyama (July 2019). "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, pp. 2623–2631.

Alim, M., G. H. Ye, P. Guan, D. S. Huang, B. S. Zhou, and W. Wu (2020). "Comparison of ARIMA model and XGBoost model for prediction of human brucellosis in mainland China: a time-series study". In: *BMJ Open* 10.12.

Altmann, Andre, Livia Toloşi, Oliver Sander, and Thomas Lengauer (2010). "Permutation importance: a corrected feature importance measure". In: *Bioinformatics* 26.10, pp. 1340–1347.

Amorim, L. B. de, G. D. Cavalcanti, and R. M. Cruz (2023). "The choice of scaling technique matters for classification performance". In: *Applied Soft Computing* 133, p. 109924.

Anand, Amber, Paul Irvine, Andy Puckett, and Kumar Venkataraman (2012). "Performance of institutional trading desks: An analysis of persistence in trading costs". In: *The Review of Financial Studies* 25.2, pp. 557–598.

Box, G. and G. Jenkins (1970). *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day.

Brewer III, E., B. A. Minton, and J. T. Moser (2000). "Interest-rate derivatives and bank lending". In: *Journal of Banking & Finance* 24.3, pp. 353–379.

Brown, Michael, Emily Chen, Sophia Lee, James Taylor, Alex Kim, and Rachel Johnson (2024). "Comparative Analysis of LSTM and Traditional Time Series Models on Oil Price Data". Unpublished manuscript.

Chen, T. and C. Guestrin (2016). "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.

Cheng, C., A. Sa-Ngasoongsong, O. Beyca, T. Le, H. Yang, Z. Kong, and S. T. Bukkapatnam (2015). "Time series forecasting for nonlinear and non-stationary processes: a review and comparative study". In: *IIE Transactions* 47.10, pp. 1053–1071.

Choi, H. K. (2018). "Stock price correlation coefficient prediction with ARIMA-LSTM hybrid model". In: *arXiv preprint arXiv:1808.01560*.

Chollet, François (2015). *Keras*. https://keras.io. Software library.

Dave, E., A. Leonardo, M. Jeanice, and N. Hanafiah (2021). "Forecasting Indonesia exports using a hybrid model ARIMA-LSTM". In: *Procedia Computer Science*. Vol. 179, pp. 480–487.

Diamond, P. (1984). "Money in search equilibrium". In: *Econometrica: Journal of the Econometric Society*, pp. 1–20.

Elsworth, Steven and Stefan Güttel (2020). *Time series forecasting using LSTM networks: A symbolic approach*. arXiv preprint. arXiv: 2003.05672 [cs.LG].

Fan, D., H. Sun, J. Yao, K. Zhang, X. Yan, and Z. Sun (2021). "Well production forecasting based on ARIMA-LSTM model considering manual operations". In: *Energy* 220.

Flavell, R. R. (2010). *Swaps and other derivatives*. Vol. 480. John Wiley & Sons.

Gers, F. A., J. Schmidhuber, and F. Cummins (2000). "Learning to forget: Continual prediction with LSTM". In: *Neural Computation* 12.10, pp. 2451–2471.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hanson, S. G., A. Shleifer, J. C. Stein, and R. W. Vishny (2015). "Banks as patient fixed-income investors". In: *Journal of Financial Economics* 117.3, pp. 449–469.

Hassani, Hossein and Mehdi R. Yeganegi (2020). "Selecting optimal lag order in Ljung–Box test". In: *Physica A: Statistical Mechanics and its Applications* 541, p. 123700.

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. With 200 full-color illustrations. New York: Springer.

Haykin, Simon (2009). *Neural Networks and Learning Machines*. 3rd. New Jersey: Pearson Education, Inc.

Hochreiter, S. (1997). "Long Short-term Memory". In: *Neural Computation*.

Hoerl, Arthur E. and Robert W. Kennard (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 12.1, pp. 55–67.

Hsieh, William W. (2004). "Nonlinear Multivariate and Time Series Analysis by Neural Network Methods". In: *Reviews of Geophysics* 42.1, RG1003.

Hull, John C. (2018). *Options, Futures, and Other Derivatives*. 10th. Pearson.

Hyndman, R. J. and A. B. Koehler (2006). "Another look at measures of forecast accuracy". In: *International Journal of Forecasting* 22.4, pp. 679–688.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: Principles and Practice*. OTexts, accessed online at https://otexts.com/fpp3/.

Jenkins, I. R., L. O. Gee, A. Knauss, H. Yin, and J. Schroeder (2018). "Accident scenario generation with recurrent neural networks". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 3340–3345.

Kallianiotis, J. N. (2013). *Foreign Currency Derivatives*.

Kandadi, T. and G. Shankarlingam (2025). *Drawbacks of LSTM Algorithm: A Case Study*. Available at SSRN 5080605.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, and T. Y. Liu (2017). "Lightgbm: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems*. Vol. 30.

Khandelwal, Pranjal (2017). *Which algorithm takes the crown: Light GBM vs XGBOOST?* Accessed: 2025-05-19. URL: https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980*.

Kontopoulou, V. I., A. D. Panagopoulos, I. Kakkos, and G. K. Matsopoulos (2023). "A review of ARIMA vs. machine learning approaches for time series forecasting in data driven networks". In: *Future Internet* 15.8, p. 255.

Lee, Hyun Dong (2019). "Innovative approach for the equity trading desk management process with the Black-Litterman model and the robust optimization". Doctoral dissertation. Stevens Institute of Technology.

Liaw, K. T. (2011). *The business of investment banking: A comprehensive overview.*

Lintner, J. (1965). "The Valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets". In: *The Review of Economics and Statistics* 47.1.

Liu, Zilong, Ruwan Godahewa, Kasun Bandara, and Christoph Bergmeir (2023). "Handling concept drift in global time series forecasting". In: *Forecasting with Artificial Intelligence: Theory and Applications*. Ed. by Robert Fildes, Fotios Petropoulos, and Vassilios Assimakopoulos. Cham: Springer Nature Switzerland, pp. 163–189.

McCulloch, Warren S. and Walter Pitts (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5, pp. 115–133.

Meisenbacher, Stefan, Michael Turowski, Kevin Phipps, Markus Rätz, Daniel Müller, Volker Hagenmeyer, and Ralf Mikut (2022). "Review of automated time series forecasting pipelines". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 12.6, e1475.

Mengle, D. (2007). "Credit derivatives: An overview". In: *Economic Review-Federal Reserve Bank of Atlanta* 92.4, p. 1.

Mochurad, L. and A. Dereviannyi (2024). "An ensemble approach integrating LSTM and ARIMA models for enhanced financial market predictions". In: *Royal Society Open Science* 11.9.

Nie, F., Z. Hu, and X. Li (2018). "An investigation for loss functions widely used in machine learning". In: *Communications in Information and Systems* 18.1, pp. 37–52.

Peedosk, Hele-Liis (2019). "Forecasting Time Series with Artificial Neural Networks". Master's thesis. University of Tartu. URL: http://hdl.handle.net/10062/64864.

Pekel, Esra and Selçuk S. Kara (2017). "A comprehensive review for artificial neural network application to public transportation". In: *Sigma Journal of Engineering and Natural Sciences* 35.1, pp. 157–179.

Raus, T (2023). *Time Series Analysis*. Course.

Said, Said E. and David A. Dickey (1984). "Testing for Unit Roots in Autoregressive-Moving Average Models of Unknown Order". In: *Biometrika* 71.3, pp. 599–607.

Schmid, Lukas, Moritz Roidl, Alexander Kirchheim, and Markus Pauly (2024). "Comparing Statistical and Machine Learning Methods for Time Series Forecasting in Data-Driven Logistics—A Simulation Study". In: *Entropy* 27.1, p. 25.

Sharpe, W. F. (1964). "Capital asset prices: A theory of market equilibrium under conditions of risk". In: *Journal of Finance* 19.3, pp. 425–442.

Siami-Namini, S. and A. S. Namin (2018). "Forecasting economics and financial time series: ARIMA vs. LSTM". In: *arXiv preprint*. eprint: arXiv:1803.06386.

Siami-Namini, S., N. Tavakoli, and A. S. Namin (2018). "A comparison of ARIMA and LSTM in forecasting time series". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. Chicago: IEEE, pp. 1394–1401.

Sirisha, U. M., M. C. Belavagi, and G. Attigeri (2022). "Profit prediction using ARIMA, SARIMA and LSTM models in time series forecasting: A comparison". In: *IEEE Access* 10, pp. 124715–124727.

Sujon, Kazi M, Roslan B Hassan, Zikri T Towshi, Mohd A Othman, Mohammad A Samad, and Keunho Choi (2024). "When to Use Standardization and Normalization: Empirical Evidence from Machine Learning Models and XAI". In: *IEEE Access*.

Sundaresan, S. (2009). *Fixed income markets and their derivatives*. Academic Press.

Szostek, K., D. Mazur, G. Drałus, and J. Kusznier (2024). "Analysis of the Effectiveness of ARIMA, SARIMA, and SVR Models in Time Series Forecasting: A Case Study of Wind Farm Energy Production". In: *Energies* 17.19.

Tashman, Larry J. (2000). "Out-of-Sample Tests of Forecasting Accuracy: An Analysis and Review". In: *International Journal of Forecasting* 16.4, pp. 437–450.

Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288.

Van Binsbergen, J. H., W. F. Diamond, and M. Grotteria (2022). "Risk-free interest rates". In: *Journal of Financial Economics* 143.1, pp. 1–29.

Watanabe, Shuhei (2023). "Tree-structured Parzen estimator: Understanding its algorithm components and their roles for better empirical performance". In: *ArXiv* abs/2304.11127. URL: https://api.semanticscholar.org/CorpusID:258291728.

Willmott, C. J. and K. Matsuura (2005). "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing avecrage model performance". In: *Climate Research* 30.1, pp. 79–82.

Wosnitzer, Robby (2016). "Mapping the trading desk: Derivative value through market making". In: *Derivatives and the Wealth of Societies*. Ed. by Dan

Bouk, Chao-Hui Jenny Lai, and Jill Lepore. University of Chicago Press, pp. 252–273.

Yankov, V. (2014). “In search of a risk-free asset”. In.

Zhang, G. P. (2003). “Time series forecasting using a hybrid ARIMA and neural network model”. In: *Neurocomputing* 50, pp. 159–175.

Zhou, Tianyi (2023). “Improved sales forecasting using trend and seasonality decomposition with LightGBM”. In: *2023 6th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE. IEEE, pp. 656–661.

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, William Vaask,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Comparative Analysis of Traditional Time Series, Machine Learning, Deep Learning and Hybrid Models for Profit Forecasting in Financial Markets, mille juhendaja on Toomas Raus, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

William Vaask
21.05.2025