MACHINE LEARNING FOR STOCK PREDICTION USING MOMENTUM STRATEGY

Master's Thesis

for acquiring the degree of Master of Science (M.Sc.)

in Information Systems
at the School of Business and Economics
of Humboldt-Universität zu Berlin

submitted by

Wing Shan Yeung

First Examiner: Prof. Dr. Stefan Lessmann
Second Examiner: Prof. Dr. Benjamin Fabian

Berlin, 28 May 2024

Contents

ABSTRACT

Forecasting stock prices are challenging in finance due to the inherent noise in stock price movements and the non-linear relationship between past performance and future stock prices. This paper aims to compare various state-of-the-art neural network models, including Long Short-Term Memory (LSTM), Temporal Convolutional Network (TCN), and transformers, for stock price prediction using the momentum strategy. The momentum strategy involves buying stocks that have recently outperformed their peers and short-selling stocks that have underperformed. Through comparative analysis across two distinct markets (S&P500 for US market and DAX for Germany), this study evaluates the effectiveness of different neural network models in forecasting future stock prices using time series data. Our study shows all models achieve similar performance by evaluating model accuracy. Moreover, our study validates the effectiveness of the momentum strategy, particularly 130-30 trading strategy, in achieving significant returns in stock prediction tasks which outperform the market portfolios.

1 Introduction

Predicting stock prices accurately is a notoriously difficult task for investors and researchers in the ever-changing world of financial markets (Enke and Thawornwong, 2005; Nti et al., 2020). Data scientists often struggle to understand the complex patterns in data over time, given the high level of noise in the stock price movement. With the evolving technology of machine learning in recent years, there has been increasing attention on its application in predicting stock market performance, as evidence has been established that machine learning techniques are capable of capturing non-linear structure in the financial market data and can effectively identify features within the noisy stock data.

In this paper, we focus on comparing different machine learning models for time series stock prediction and trying to generate profit using momentum strategy. The momentum strategy is a trading strategy based on the relative performance of stocks against each other, and was first proposed by Jegadeesh and Titman (1993) in the paper "Returns to buying winners and selling losers: Implications for stock market efficiency". The U.S. stocks that perform the best over a three to 12-month period tend to continue to outperform on average over the next few months, and vice versa. By buying stocks with good past performance (winners) and selling stocks that are underperforming (losers), significant returns can be generated (Takeuchi and Lee, 2013; Krauss et al., 2017; Fischer and Krauss, 2018).

Past studies have explored various machine learning models like stacked Boltzmann machines (Takeuchi and Lee, 2013), random forest (Krauss et al., 2017; Zhang et al., 2022), and LSTM (Fischer and Krauss, 2018; Fjellström, 2022) for predicting stock prices using the momentum strategy, all yielding notable positive return compared to the market performance.

Building upon the achievements of prior papers in integrating machine learning with the momentum strategy, we would like to explore various neural network architectures for forecasting time series stock movements within the momentum strategy framework.

Building upon these findings and the methodology outlined in Krauss et al. (2017), Fischer and Krauss (2018) and Takeuchi and Lee (2013), our focus is on employing state-of-the-art neural network models, specifically long-short-term memory (LSTM), temporal convolutional network (TCN), and transformers (TF), and comparing their results to baseline models, including random forest (TF) and feedforward networks (FNN). Our goal is to investigate whether deep learning techniques can effectively identify features within time series stock prices using data from relatively recent years (2010-2017), ultimately aiming to generate profitable returns. We aim to address three research questions: first, which machine learning algorithm best predicts stock prices? Second, will the performance of the machine learning models align across different markets? Lastly, can we generate higher profits using the momentum strategy with different long-short trading strategies?

Given the gap in existing literature, our contribution is threefold. First, we compare the state-of-art neural networks, including LSTM, transformer, and TCN, which have been proven suitable for time series prediction but have not been tested against the time series stock prediction using momentum strategy. Second, we would like to compare the models in different stock markets, including S&P500 for the US stock market and DAX for the German market, to validate the performance of the models across various marketplaces. Lastly, we evaluate our return with different long-short trading strategies, including equal-weight and 130/30 strategies.

The rest of the paper is organized as follows. Section 2 presents the literature review for machine learning in time series stock prediction, momentum strategy and related work. In section 3, we present our methodology and framework. Then, we explain our results and findings in section 4. Lastly, we conclude our findings in section 5 and the limitations of our study and potential directions for future work in section 6.

2 Literature Review

Accurate forecasting is essential for the finance field when making investment decisions. It is difficult to accurately predict the finance data, as the predictive signals (features) and future return (target) are not in a linear relationship (Fischer and Krauss, 2018). In fact, stock markets are affected by numerous highly interrelated variables, such as general economic conditions, social factors, political events at both homegrown and international levels, and even natural events (Enke and Thawornwong, 2005; Nti et al., 2020). The realm of financial forecasting is marked by an abundance of data, noise, non-stationary, unstructured nature, as well as high levels of uncertainty and hidden relationships, making it challenging to accurately predict the future return (Bao et al., 2004).

According to the efficient market hypothesis, stocks always trade at their fair value on exchanges, making it impossible for investors to purchase undervalued stocks or sell stocks for inflated prices (Yen and Lee, 2008). Therefore, it should be impossible to outperform the overall market through expert stock selection or market timing, and the only way an investor can obtain higher returns is by purchasing riskier investments. However, numerous instances of irrational behaviour have been identified, including overreaction, the disposition effect (the tendency to sell previously purchased stocks that have appreciated and the reluctance to sell those that are trading below their purchase price) and the weekend effect (the stock returns on Mondays are significantly lower than other four days) (Hsu et al., 2016). Although supporters of the efficient market hypothesis argue against the feasibility of developing any predictive framework capable of accurately forecasting stock price movements, notable literature has showcased instances where stock price time series can be predicted with remarkable precision (Lewellen, 2002; Schwartz and Whitcomb, 1977). In addition, Lo et al. (2000) prove that technical analysis can be used to predict stock movements.

Due to the uncertainty and volatility of the stock markets, investments in the stock markets are often guided by some form of prediction. Prediction methodologies are commonly categorized as fundamental analysis and technical analysis (charting) (Attigeri et al., 2015; Nti et al., 2020; Ican et al., 2017; Li et al., 2015). The fundamental analysis approach focuses on evaluating the underlying company rather than the stock itself. The input data are qualitative data which is unstructured, such as news, social media, etc, which poses a challenge to the prediction. On the other side, the technical analysis predicts the future price of stock by studying the historical trends of the stock price and the technical indicators. Examples of quantitative (structured) data are the historical stock price, including previous closing price, opening price, price change, volume, etc.

Inspired by the rapid evolution and breakthroughs observed in neural network development over recent years, alongside the documented successes in technical analysis outlined in prior research (Li and Bastos, 2020) and recognizing the structured nature of historical stock price data, our study is focused mainly on the domain of technical analysis using quantitative data. Specifically, we focus on time series stock prediction methodologies. Time series data is "a well-ordered arrangement of data or a set of data points that a variable takes at equally placed time intervals" (Idrees et al., 2019), which is "often the result of the combined effects of several different factors" (Cao and Wang, 2019). Financial time series is "a series of historical data generated in economic activities arranged according to the sequence

of occurrence time" (Cao and Wang, 2019). In conclusion, we want to make an estimate of the future based on past observations in the finance market.

2.1 Machine Learning for Time Series Data Prediction

Forecasting in the financial time series involves forecasting the next step in the series by considering various relevant variables (Ican et al., 2017). This task is crucial for investors, traders and financial analysts who rely on accurate predictions to make informed decisions in volatile markets. However, humans often struggle to handle the complexities and uncertainties inherent in financial data, leading to challenges in accurately predicting the stock market. With the emergence of data science as a powerful tool for data analysis, various machine learning models have been developed or extended to uncover hidden patterns, detect trends and make predictions based on historical data, resulting in more precise finance forecasting.

Under the machine learning category, the time series stock market prediction techniques can be categorized into two types, namely non-supervised (clustering) and supervised (regression and classification) (Gandhmal and Kumar, 2019; Kumbure et al., 2022). Clustering-based techniques (non-supervised learning), e.g. filtering, fuzzy k-means, are usually used to cluster similar stocks together in the first stage and apply clustering-based prediction model to forecast the future stock price (Vilela et al., 2019; Zhong and Enke, 2017a). Supervised machine learning, e.g. random forest, neural network, can be further broken into two categories, namely regression and classification (Kumbure et al., 2022). Various machine learning models have been developed or adapted specifically for the supervised machine learning tasks. These models range from traditional statistical methods like autoregressive integrated moving average (ARIMA) to more complex algorithms such as support vector machines (SVM), random forests and deep learning architectures like recurrent neural networks (RNNs), long-short-term memory networks (LSTMs), temporal convolutions networks (TCN) and transformers (TF). As our goal is to predict the stock price in this paper, we focus on supervised learning.

Table 1 categorized the literature by the type of problem and their prediction target. The additional details of the literature, including the dataset used, input variables, included models, evaluation metrics, and the best-performing models, have been summarized in Appendix A.

2.1.1 Box-Jenkins models

One of the most commonly used approaches for time series forecasting is Box-Jenkins models, such as autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) (Makridakis and Hibon, 1997). By combining an autoregressive (AR) process and moving average (MA) processes, these models capture both the linear dependence on past observations and the residual errors to forecast time series data. The future value of a variable is a linear combination of past values and past errors. The differentiating factor is that the ARIMA model adjusts non-stationary data into a stationary format prior to its analysis. Due to its ability to adapt to non-stationary data, these Box-Jenkins models are widely adopted in stock price prediction.

Previous researches have proven that the ARIMA model is useful in time series finance prediction and able to predict the future stock price with low error rate (e.g. MSE or MAE) (Adebayo et al., 2014; Reddy, 2019). Ariyo et al. (2014) present the extensive process of building a stock price predictive model using the ARIMA model using published stock data obtained from the New York Stock Exchange (NYSE) and the Nigeria Stock Exchange (NSE). Their result revealed that the ARIMA model has a strong potential for short-term prediction and can perform favourably with existing techniques for stock price prediction (Ariyo et al., 2014).

Even though ARIMA models are very prevalent in modelling economic and financial time series, it has an obvious limitation of linearity. For instance, it is hard to model the nonlinear relationships between variables in a simple ARIMA model. Furthermore, it is assumed that there is a constant standard deviation in errors in the ARIMA model, which, in practice may not be satisfied. (Reddy, 2019) check the stationarity in time series data and predict the direction of change in the stock market index (BSE and NSE) using the stochastic time series ARIMA modelling . Their inputs

Type of Problem	Target	Authors
Classification Task	Direction of Stocks	Ariyo et al. (2014) Kara et al. (2011) Khaidem et al. (2016) Dey et al. (2016) Zhong and Enke (2017b) Shen et al. (2012) Kara et al. (2011) Kim and Kang (2019) Li et al. (2022) Tino et al. (2001) Kim (2003)
	Whether a stock is outperforming the market median	Takeuchi and Lee (2013) Krauss et al. (2017) Fischer and Krauss (2018) Fjellström (2022) Zhang et al. (2022) Xiang and Wang (2023)
	Different price change segments' probability	Dai et al. (2022)
Regression Task	Future Stock Price	Reddy (2019) Bao et al. (2004) Rady et al. (2021) Karmiani et al. (2019) Selvin et al. (2017) Wang et al. (2022) Hu (2021) Siami-Namini et al. (2018) Althelaya et al. (2018) Sethia and Raut (2019) Cao and Wang (2019)

Table 1: Classification of Stock Time Series Predictive Tasks in Literature

are the weekly closing index values of BSE and NSE. By testing different setups of the ARIMA models, The result confirmed the ability of the ARIMA model to forecast the future time series in the short run. In the long run, there is more noise and nonlinearity in the data due to changes in government policies or economic instability. Thus, ARIMA models are not able to capture the accurate trend.

2.1.2 Support Vector Models

The effectiveness of linear models hinges on the assumption that the underlying data-generating process follows a linear pattern and the data generated is not random. In traditional financial economics, there's a perspective that market prices behave randomly, making past prices unreliable indicators of future price movements. In the course of growing

instability in financial markets, attention to chaos theory is growing (Klioutchnikov et al., 2017; Bao et al., 2004). Chaos theory proposes that what appears random could stem from a deterministic function that isn't random (Chavas and Holt, 1993). In such scenarios, Box-Jenkins models lose their edge, and a different machine learning algorithm for finance prediction is needed.

Support vector models (SVM) is a supervised machine learning model that can be used for classification and regression analysis. The core concept of SVM is to find the optimal hyperplane that best separates the data into different classes. This hyperplane is positioned to maximise the margin, which is the distance between the hyperplane and the closest data points from each class, known as support vectors. By maximizing the margin, SVM aims to achieve the best generalization performance, thus reducing the risk of overfitting. It also employs the kernel trick, which involves mapping the original input data into a higher-dimensional space where it becomes linearly separable Fletcher (2009).

SVMs have been utilized for classification as well as regression problems. Bao et al. (2004) implement SVM regression in forecasting stock price by the daily closing price of Haier (an electrical manufacturer in China) from April 2003 to Nov 2003, and show that SVM is a promising alternative to financial time series forecasting with three benefits. Firstly, SVMs adhere to the principle of structural risk minimization, which seeks to minimize an upper bound of the generalization error rather than solely focusing on minimizing the training error. Secondly, SVMs require relatively few controlling parameters compared to other machine learning algorithms. Lastly, SVMs provide a globally unique solution derived from quadratic programming techniques, ensuring consistency and robustness in model optimization.

Instead of regression, Kara et al. (2011) approach the prediction problem as a classification task, using artificial neural networks (ANN) and support vector machines (SVM) to predict the directional movement of daily stock price indices. They utilize datasets from the KSE-100, KOSPI, Nikkei 225, and SZSE indices spanning from 2011 to 2020. Technical indicators such as closing, opening, daily high, and daily low prices are used as input features to construct single-layer ANN and SVM models with linear, radial basis function (RBF) and polynomial kernels. They conclude that both the ANN and SVM models are useful tools to predict the directional movement of the stocks, but performance evaluation based on accuracy and F-score from the confusion matrix reveals that ANN outperforms the SVM model due to the complexity of the stock market.

Shen et al. (2012) also apply SVM to predict the next day's directional movement of three different indices (NASDAQ, S&P 500, and DJIA) using daily prices from Jan-2000 to Oct-2012. They use the world's major stock indices, including the stock index (NASDAQ, Nikkei 225), currency (EUR, AUD, JPY, USD), and commodity (silver, oil, platinum, gold), as input features for their machine learning-based prediction. They apply SVM and MART for the prediction, and both models achieve a high prediction accuracy of 70~75%. With a longer time window size, SVM performs better. This is because the longer the period is, the more information is available and the higher resistance of their prediction to noise. However, they also mention that the SVM algorithm is very sensitive to the size of training data. When the size of the training set is not large enough, the SVM algorithm might not be able to split the data properly with the hyper-plan. Moreover, feature selection is essential when using SVM to generate a better result.

2.1.3 Tree-based Models

A decision tree, or a classification and regression tree (CART) is a simple yet powerful machine-learning algorithm used for both classification and regression tasks. As the name suggested, CART is straightforward, non-parametric and can be used in either for classification or regression task (Rigatti, 2017). It partitions the input space into regions, making decisions based on a sequence of binary splits. At each node of the tree, a decision is made by selecting the optimal decision threshold for a variable based on recursive partitioning, in another word, testing the potential value of different thresholds for each predictor and implementing the most valuable until further splitting no longer improves discrimination (Song and Ying, 2015; Yamada et al., 2003). Decision trees can handle non-linear relationships between the input variables and the target variables. As time series data often have complex patterns and non-linearity, these tree-based models can perform well with the time series prediction tasks.

However, single decision trees are not suitable for analyzing complex data with many interacting predictor (Rigatti, 2017). In order to improve the performance of decision trees, they are usually combined with ensembling learning, including bagging, stacking and boosting. The ensemble method is an information mining approach which combines multiple learning algorithms in order to obtain better predictive performance (Parmar et al., 2019). The most popular ensemble tree models are gradient-boosted trees and random forests.

Random Forest Random forest is a tree-based algorithm that extends bagging with random subspace. In the random forest algorithm, we first use bootstrapping to randomly select a set of samples from the original data set with replacement (Biau and Scornet, 2016). The process of aggregating a new sample is called "bagging". Another randomization occurs at the decision nodes. For each individual tree, we draw a random sample of attributes instead of searching among all attributes. By ensembling decision trees and adding randomization to the inputs, the random forest has a lower variance and is able to produce better results compared to using a single tree (Rigatti, 2017). Same as decision tree, random forest can be used on both regression tasks and classification tasks. With the regression task, the estimate is the mean forecast of all the decision trees. With the classification task, the forecast value is that class label that has received the majority of votes across all the decision trees of the ensemble.

Gradient boosting tree The Gradient boosting tree is a tree-based machine learning technique that combines the principles of boosting and decision trees. Boosting is an ensembling learning method which built on the principle that a collection of weak learners can be combined to produce a strong learner (Friedman, 2002). Here, a weak learner is a tree that achieves results slightly superior to a random guessing model, while a strong learner is defined as a hypothesis with exceedingly high accuracy. Gradient boosting tree starts with a simple decision tree, which do not perform well and have a high level of errors. Built upon the mistake of previous tree, a new model will learn from the previous error and update the prediction. The process to build more weak learns is iterated until the model prediction stops improving. Lastly, all these base model predictions is weighted to create a single strong model.

Previous researches have proven that tree-based models works on time series prediction. Rady et al. (2021) compare different tree-based methods for time series forecasting. They compare the performance of the decision tree, random forest regressor, gradient-boosted tree and ARIMA model on monthly gold prices from Nov 1989 to Dec 2019. With the the monthly gold price, there are 362 observations in total as input variables. They then train the models on the training set and evaluate the model performance using root-mean-square error (RMSE). The result shows that RF has the lowest prediction error (RMSE) compared to other models.

Although the tree based models are commonly used on time series related tasks, they lacks inherent memory like neural network. Hence, the step to convert time series data to supervised learning data is needed. This involves reconstructing the sequential nature of the data into input features, such as extracting technical indicators (Khaidem et al., 2016; Dey et al., 2016) or converting the daily stock return into cumulative return (Krauss et al., 2017), thereby converting the time series prediction task into a supervised machine learning problem.

Khaidem et al. (2016) apply random forest for predicting the direction of stock price movement for APPL, MSFT, and Samsung. By treating the forecasting problem as a classification problem, the forecasting error can be minimized. To predict the stock price behaviour in the future, they first acquire past stock price data, smooth and extract technical indicators, such as the relative strength index as a momentum indicator, percentage change for the price, trading volume, etc. With the random forest, they successfully predict the direction of stocks with 85% accuracy and prove random forest is very robust in predicting the direction of stock price movement.

Dey et al. (2016) predict the direction of the stock market price using extreme gradient boosting (XGBoost) using the Apple Inc. data set. They derive features from a dataset comprising closing price, opening price, high and low prices, and trading volume, which are then transformed into technical indicators like relative strength index, Williams %R, rate of change, and trading volume, among others. They then apply XGBoost and the result shows that the XGBoost performs better than ANN, RF, SVM and logistic regression.

Krauss et al. (2017) compare the performance of Deep neural networks, gradient-boosted trees and random forests on the S&P500 index on predicting whether a stock will outperform cross-sectional market median on the next day. For the tree-based models, they use the past 240 days' daily return of stocks as input features and convert the daily return into the monthly cumulative return. The result shows that random forest outperforms gradient-boosted trees and deep neural networks.

2.1.4 Neural Network Models

Artificial Neural Networks Neural network models for finance forecasting have also gained popularity over the years. Artificial neural networks (ANNs) are the most basic form of neural network model designed to mimic the structure of the human brain. They consist predominantly of numerous interconnected computational nodes, known as neurons, which collaborate in a distributed manner to collectively learn from the input data and optimize their final output (O'shea and Nash, 2015). ANNs can capture the structural relationship between a stock's performance and its determinant factors more accurately than many other statistical methods (Gandhmal and Kumar, 2019; Selvin et al., 2017). A basic form of neural network is the Feedforward neural network (FNN). FNN shares the same architecture as ANN, which consists of three types of layers: the input layer, hidden layers and the output layer. In this network, information moves in only one direction forward from the input nodes, through the hidden nodes and to the output nodes. There is no feedback from the outputs of the neurons towards the inputs throughout the network (Bebis and Georgiopoulos, 1994). Inputs are directly connected to the outputs through a sequence of weights, making it a simple and straightforward architecture. Depending on the number of layers, all neural network architectures can either be classified as "single layer" or "multi-layer" (Deboeck, 1994). In a single-layer feed-forward neural network, there exists only an input and output layer exists. Input signals are passed on to the output layer via the weights, and the neurons in the output layer compute the output signals. In a multi-layer feed-forward neural network, there is at least one layer of hidden layer between the input and output layers, which can be considered as the computational engine of the neural network. The function is to "intervene between the external input and the network output in some useful manner". Each hidden layer's neurons take the weighted sum of the outputs from the previous layer, apply an activation function and pass the result to the next layer, enabling the neural network to extract higher-order statistics.

Kurani et al. (2023) provide a comprehensive comparative study on SVM and ANN in stock market forecasting. They conclude that SVM and ANN are among the most feasible techniques that can be used to achieve greater accuracy in predicting stocks. Using SVM offers the advantage of reducing the burden of comparing present price patterns with historical ones, while the core benefit of ANN lies in its ability to discover non-linear relationships in an input dataset without prior knowledge of the dependency between input and output.

Zhong and Enke (2017b) apply Artificial Neural Networks (ANNs) with dimension reduction techniques, including Principal Component Analysis (PCA), Fuzzy Robust Principal Component Analysis (FRPCA), and Kernel-based Principal Component Analysis (KPCA), to predict the daily direction of the closing price for the S&P 500 ETF. It is essential for ANNs to achieve accurate results by deliberately selecting input variables and finding an optimal combination of network parameters. The research finds that ANN-PCA performs slightly better than the other two in terms of prediction accuracy and risk-adjusted profits. Another variety is deep neural networks, which are based on ANNs but have more layers between the input and output layers.

In Kara et al. (2011), both ANN and SVM are used to predict the direction of stock price movement in the Istanbul Stock Exchange. Both models demonstrate significant performance in predicting the direction of stock price movement, with the average prediction performance of the ANN model found to be significantly better than that of the SVM model.

Kim (2003) applies SVM to forecast the direction of daily price change in the Korea Composite Stock Price Index (KOSPI) and compares it with back-propagation neural networks and case-based reasoning (CBR). The results show that SVM performs better. However, she also observes that the prediction performance of SVMs are sensitive to the

value of parameters, including the upper bound and the kernel parameter, and concludes that it is important to find the optimal value of these parameters.

Recurrent Neural Network Recurrent Neural Networks (RNNs) are deep learning architectures specifically designed to process sequential data, such as time series, text, and speech. Unlike feedforward neural networks, which process each input independently, RNNs possess an internal memory that allows them to retain information about previous inputs. RNNs take prior inputs to influence the current input and output, while the traditional deep neural networks assume that inputs and outputs are independent of each other (Elman, 1990). This memory mechanism enables RNNs to capture temporal dependencies within the data, making them particularly well-suited for tasks involving sequential information. One of the key features of RNNs is their ability to handle inputs of variable lengths, making them versatile for a wide range of applications.

Tino et al. (2001) apply RNN to predict the next day's stock price for the DAX index and FTSE 100 using the previous day's closing prices and the corresponding call and put options. As RNNs tend to either overestimate noisy data or behave like finite-memory sources with relatively shallow memory, they applied the Elman Recurrent Neural Network. This model combines sophisticated models fitted on a larger dataset with a fixed set of simple-minded symbolic predictors using only recent inputs. They compare the results with a Markov model and the GRACH family. With careful use of recurrent networks and Markov models, they are able to generate a statistically significant excess profit. The authors argue that there is no reason to prefer RNNs over the more simple and straightforward Markov models.

However, traditional RNNs suffer from the vanishing gradient problem, which limits their ability to capture long-range dependencies in sequences (Hochreiter and Schmidhuber, 1997; Graves and Graves, 2012). The vanishing gradient problem means that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections. If the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state. To address this limitation, variants of RNNs such as Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) have been developed. These architectures incorporate gating mechanisms that allow them to selectively update and forget information over time, making them more effective at capturing long-term dependencies. As a result, RNNs, particularly LSTM and GRU variants, have achieved excellence performance in various sequential data tasks, including natural language processing, speech recognition, and time series forecasting.

Long Short-Term Memory (LSTM) is a Recurrent Neural Network (RNN)-based model commonly used in time series forecasting. Unlike traditional RNNs, which may struggle to retain information over long sequences due to the vanishing gradient problem, LSTMs are equipped with specialized memory cells that can maintain information over extended time periods. The key components of an LSTM cell include the forget gate, input gate, and output gate, which allow much more information to enter, leave, or be forgotten. The forget gate decides whether to keep the information from the previous time step or forget it. Subsequently, the input gate is used to quantify the importance of the new information carried by the input. Lastly, the output gate turns the useful information from the previous cell state and presents it as output. With this architecture, LSTMs can handle long-term dependencies and prevent the vanishing gradient problem. Thus, LSTM is a popular model for time series data, as it is able to learn long-term dependencies and avoid the vanishing gradient problem.

LSTM performs well with the learning capacity, but the computational cost also increased significantly. Therefore, Gated Recurrent Unit (GRU) was introduced by Cho et al. (2014) to address the computational burden. GRU's architecture is similar to LSTM as it also works to address the short-term memory problem of RNN models. Instead of using a "cell state" to regulate information, it uses hidden states, and instead of three gates, it has two gates, which are a reset gate and an update gate, resulting in fewer parameters than LSTMs. Similar to the gates within LSTMs, the reset and update gates control how much and which information to retain. As there are fewer parameters than LSTMs, GRUs are often faster to train than LSTMs, but they may not be as effective for learning long-range dependencies.

Karmiani et al. (2019) compare SVM, LSTM, and back-propagation models to predict stock prices for different technology companies, such as Apple, Amazon, Google, etc., from Jan 2009 to Oct 2018. The input features include the historical performance of stocks, including high, low, close, and volume. LSTM performs the best among all three models and is the most reliable model based on a t-test despite being comparatively slow. Kim & Kim (2019) introduced a hybrid LSTM-CNN model for forecasting stock prices. They evaluated the model using S&P 500 ETF data spanning from October 2016 to October 2017, incorporating historical stock performance like high, low, open, close price, and volume. Their findings demonstrated that the ensemble model outperformed both individual LSTM and CNN models (Kim and Kim, 2019).

Kim and Kang (2019) conduct a comparative analysis of several deep learning models, including multi-layer perceptron (MLP), one-dimensional convolutional neural networks (1D CNN), stacked long short-term memory (stacked LSTM), attention networks, and weighted attention networks, for predicting financial time series using the KOSPI 200 index and other indices, including currency, commodities, and global indices. They utilized the ratio of change for the KOSPI 200 index with varying lookback periods (i.e. 5, 10, 15, 20, 30, 60 days) to forecast the direction of change on the next day. Results indicated that a longer lookback period resulted in higher hit ratios across all models, regardless of the model. All models showed a high hit ratio, and the attention LSTM model outperformed the others with a 60-lookback period, in other words, longer sequential data.

Siami-Namini et al. (2018) apply ARIMA and LSTM models to predict different stock prices and compared their performance in reducing error rates. They used different adjusted close prices of historical monthly index prices (including N225, NASDAQ composite index, etc.) and historical monthly commodities prices (such as housing, food and beverage, etc.) from Jan 1985 to Aug 2018 as inputs. The results showed that the performance of LSTM is superior compared to ARIMA, with an average reduction in error rates of 84-87%. They also reported that when the number of epochs changed, there was no improvement in performance.

Althelaya et al. (2018) investigate and compare the application of different RNNs into the finance time series prediction. They use the multivariate historical data for the S&P 500 index, including the close price, open price, low price, high price and volume, for the period from January 2010 to November 2017 as input. They apply RNN, bidirectional LSTM, stacked LSTM, bidirectional GRU and stacked GRU to predict both long and short-term prices of the S&P500 index. The result shows that the stacked LSTM architecture has demonstrated the highest forecasting performance for both the short and long term.

Sethia and Raut (2019) compare the performance of LSTM, GRU, ANN and SVM models for the prediction of the S&P 500 index. They first obtain the S&P 500 historical index data, including open, high, low, close and volume, from 2000 to 2017. Then, fifty technical indicator-based attributes, such as daily returns, 24-day simple and exponential moving averages, etc, were calculated. They conduct feature extraction using PCA and reduce the number of features to 12. Afterwards, they apply different models to predict whether the price five days later is higher than the current price. If so, a buy signal is generated, and one share of that stock is purchased on the current date and sold on the 5th day. The LSTM model proved to outperform the other models with a return of 400% greater than the hold and wait strategy and an R2 score of 0.9486.

Convolutions Neural Networks Besides RNNs, Convolutional Neural Networks (CNNs) are also popular models. CNNs are a type of regularized multi-layer feedforward network which learns feature engineering by itself through filter operations (O'shea and Nash, 2015). Apart from input and output layers, a CNN typically comprises three layers: a convolutional layer, a pooling layer, and a fully connected layer. Convolutional layers are responsible for feature extraction, systematically convolving across the entire input and responding exclusively to data stimuli within the currently focused window. Through this sliding filter or kernel operation, CNNs extract features simultaneously, such as edges in image processing tasks or peaks/lows in time series data. Subsequently, this process reduces the size of the data. Pooling layers are then used to reduce the spatial dimensions of the feature maps, while fully connected layers perform the final classification or regression tasks based on the extracted features. In the fully connected layers,

once the feature maps are sufficiently abstracted, fully connected layers are used for classification or regression tasks. These layers connect every neuron in one layer to every neuron in the next layer, enabling the network to learn complex relationships between features and make predictions. The final fully connected layer typically produces the output of the network, representing the class probabilities in the case of classification tasks or the predicted values in regression tasks. Compared to RNNs, such as LSTM, CNNs have the strength of faster training with simple structures and no dependency constraints on previous steps (Kirisci and Cagcag Yolcu, 2022). They focus on some key features of the series from a global perspective.

Although CNNs are mostly applied for analyzing images, there is evidence to show that CNN architectures are also capable of identifying changes in trends and are suitable for time series data (Selvin et al., 2017). Selvin et al. (2017) use LSTM, RNN, and CNN models with a sliding window approach to predict stock prices for NSE (National Stock Exchange) listed companies for the period July 2014 to June 2015. The dataset contains minute-wise data from NSE-listed companies. The window size was fixed to be 100 minutes with an overlap of 90 minutes of information, and predictions were made for 10 minutes in the future. CNN captures these changes more accurately compared to the other two models in the specific setup. Compared to LSTM and RNN, CNN architectures mainly focus on the given input sequence and do not use any previous history or information during the learning process. RNN and LSTM also take into account the information from previous lags, hence resulting in better performance compared to CNN.

Cao and Wang (2019) propose the use of a hybrid prediction model based on CNN and SVM to forecast financial stock prices for five different indexes, including HIS, TSEC, DAX, NASDAQ, and S&P 500, representing three financial markets in Asia, Europe, and the US. After testing different hyperparameters to determine the best-performing CNN, they compared the results with those of CNN alone. They concluded that CNN can effectively handle continuous prediction variables and yield good prediction results. Moreover, their proposed model can further enhance the performance of CNN.

A specific type of time series CNN is the Temporal Convolutional Network (TCN). Bai, Kolter, & Koltun (2018) (Bai et al., 2018) proposed TCN for sequence modelling tasks. They prove that TCN performs much better than generic RNNs (such as LSTM and GRU) in sequence modelling tasks in terms of accuracy, a longer memory, and a smaller model size, which makes it suitable for dealing with financial time series data. TCN is a combination of a 1D fully-convolutional network and causal dilated convolutions, which allow input of any length and produce the same length of output. Causal convolutions are applied, where output at time t is convolved only with elements from time t and earlier in the previous layer in order to avoid leakage by using future information. Employing the dilated layer enables an exponentially large receptive field instead of a size linear in the depth of the network, resulting in much deeper and more effective use of memory. Residual blocks are then added to the layer for activation, normalization, and regularization.

Dai et al. (2022) compare TCN with GARCH family models and LSTM models for predicting price changes in the Chinese Shenzhen Stock Exchange 100 Index (SZSE 100) in their study. They categorize the price changes into several categories and deploy conditional probability for each category using TCN, TCN (attention), GARCH family, such as SGARCH-norm, SGARCH-std, and LSTM. TCN (attention) is the TCN with an attention mechanism layer, which assigns higher weights to important features, thus improving the effectiveness of the model. The results show that both TCN and TCN (attention) models exhibit better overall performance than the other models. TCN(attention) and TCN perform similarly well.

Xiang and Wang (2023) predict the intraday trading direction movement of CSI using the TCN model and compare the performance of TCN with other machine learning models, such as LSTM and RF. They collect the adjusted closing and opening prices for all stocks in the CSI index from 2005 to 2020 and generate intraday returns, returns in relation to the last closing price and returns in relation to the opening price as the input features for RF input. For TCN and LSTM, they input 240-time steps and three features. The models are then employed to predict the probability of whether each

stock will outperform the cross-sectional median of future periods. Their results demonstrate the TCN cumulative returns outperformed LSTM and RF.

Transformers Transformers (TF) have also gained popularity over recent years. They were first proposed by Vaswani et al. (2017) in the paper "Attention is All You Need" , which showed that transformers generalize well for sequence-to-sequence tasks, such as translation, and are trained significantly faster than recurrent or convolution-based models. Compared to complex recurrent or convolutional neural networks, transformers are relatively simple network architectures based entirely on attention mechanisms. In a transformer architecture, there are no recurrent computations to raise scalability. Instead, transformer blocks are used.

The transformer block consists of an encoder-decoder architecture, with multiple layers of self-attention mechanisms. The encoder processes the input sequence to create a representation, while the decoder generates the output sequence based on this representation. Each layer in the Transformer contains multiple self-attention sub-layers and feed-forward neural networks, with residual connections and layer normalization applied after each sub-layer. The self-attention mechanism allows the model to weigh the importance of different words in the input sequence, enabling it to capture long-range dependencies and relationships more effectively. This attention mechanism is computed using scaled dot-product attention, where each word in the sequence attends to every other word, and the attention weights are computed based on their relevance.

Wang et al. (2022) implement several back-testing experiments on the main stock market indices worldwide, including CSI 300, S&P 500, Hang Seng Index, and Nikkei 225. They predict the stock closing price with the previous days' normalized closing prices. All the experiments demonstrate that Transformer outperforms significantly compared to other models, such as RNN and LSTM, and can generate excess earnings for investors. They suggest that the movements of stock indices are highly noisy. Thus, with the encoder-decoder architecture and the multi-head attention mechanism, Transformer can better characterize the underlying rules of stock market dynamics.

Li et al. (2022) proposed a transformer-based attention network framework for stock movement prediction. Using historical daily stock data of 5 days' daily prices and tweets, they predict the direction of stock movement for the target trading day. The results show that the proposed framework outperforms other models, including ARIMA, CH-RNN (attention-based RNN), and Adv-LSTM (attention-based LSTM), with an accuracy of 56%.

The Temporal Fusion Transformer (TFT) is an extension of the Transformer architecture designed to adopt the unique characteristics of time series data, proposed by (Lim et al., 2021). TFT is an attention-based DNN architecture composed of static covariate encoders, gating mechanisms, sequence-to-sequence layers, and temporal self-attention decoders. In their paper, the significant forecasting performance of TFT is demonstrated by evaluating it across four distinct time-series datasets: electricity, traffic, retail, and volatility.

One of the main features of TFT is its ability to effectively capture temporal patterns and dependencies present in sequential data. This is achieved through the use of self-attention mechanisms, allowing the model to weigh the importance of different time steps in the input sequence when making predictions. By attending to relevant time steps, TFT can effectively capture both short-term and long-term dependencies in the data. Another important aspect of TFT is its autoregressive structure, enabling it to generate predictions recursively by feeding its own predictions back into the model. This enables TFT to capture dynamic patterns and adjust its predictions based on previously generated forecasts.

Hu (2021) applies a temporal fusion transformer for stock price prediction using previous stock prices, including the high, low, open, and close prices. The dataset includes the entire S&P 500 companies' stock trends over the last ten years, from January 4th, 2010 to May 31st, 2021. They compare the results with Support Vector Regression (SVR) and Long Short-Term Memory (LSTM) and find that TFT achieved the lowest error.

In summary, leveraging advanced machine learning frameworks for financial data analysis has proven effective in improving stock prediction accuracy. Each model has its strengths and weaknesses, but neural network models, particularly RNNs, TCNs, and transformers, have emerged as a leading approach due to their ability to capture complex

patterns and relationships in financial data. Their flexibility and scalability further enhance their utility in adapting to diverse market conditions and datasets, making them invaluable tools for stock prediction tasks.

2.2 Momentum Trading Strategies with Machine Learning

Price momentum is a trading strategy based on the relative performance of securities against each other and was first proposed by Jegadeesh and Titman (1993). Based on their empirical findings, securities that recently outperformed their peers over the past 3 to 12 months will continue to outperform on average over the next few months. By buying stocks with good past performance (winners) and selling stocks that are underperforming (losers), they are able to generate significant abnormal returns over the 1965 to 1989 period. Subsequent studies again prove the momentum strategy to be profitable over different time periods (Chan et al., 1999; Jegadeesh and Titman, 2001, 2023) and over different countries and in most large markets (Rouwenhorst, 1998). A notable exception mentioned in a follow-up study by Jegadeesh & Titman (2023) is that the evidence supporting momentum in Asia is relatively weak (Jegadeesh and Titman, 2023).

In order to generate profit through the momentum strategy, it is important to predict the winners and losers accurately. Previous literature highlights the superiority of classification models over regression models in forecasting financial market data and investment returns (Leung et al., 2000), prompting our focus on constructing a classification problem. There are several related previous researches which provide the initial applications of machine learning techniques related to the momentum strategy, including Huck (2009), Moritz and Zimmermann (2016), Takeuchi and Lee (2013), Krauss et al. (2017) and Fischer and Krauss (2018).

Huck (2009) first proposes a framework for pairs selection that utilizes ELECTRE III multiple return forecasts based on bivariate information sets and multi-criteria decision techniques that output potentially undervalued and overvalued stocks by ranking stocks based on the anticipated returns. The methodology involves three key steps: forecasting, outranking, and trading. During forecasting, Huck applies Elman neural networks to generate one-week-ahead return forecasts using the time series data from previous date. In the outranking step, with the construction of a square decision matrix of anticipated spreads based on the different return forecasts, he ranks the different stocks in terms of anticipated returns. In the trading step, the top k stocks of the ranking are bought, and the bottom k stocks are sold short. He tests the methodology on S&P 100 constituents from 1992 to 2006, and the weekly returns exceed more than 0.8 percentage with 54 percent directional accuracy for k=5.

Takeuchi and Lee (2013) explore the integration of a feedforward neural network model using an autoencoder composed of stacked restricted Boltzmann machines to extract features from the history of individual stock prices . The model is able to discover an enhanced version of the momentum effect in stocks without extensive hand-engineering of input features and generate high profits over the 1990-2009 test period. Specifically, they use the 12 monthly returns and 20 daily returns as input variables to predict whether the stock is going to outperform the cross-sectional median of all stocks for each month or day. Then they rank all stocks each month by the probability of outperforming the cross-section median, buy those in the top decile, and sell those in the bottom decile. With an overall accuracy of 53.84%, their strategy delivers an annualized return of 45.93% over the 1990-2009 test period versus 10.53% for basic momentum.

Moritz and Zimmermann (2016) introduce a tree-based conditional portfolio sorting framework by deploying random forests using U.S. CRSP data from 1968 to 2012. Specifically, they construct a set of decile rankings for the non-overlapping one-month returns over the two years before portfolio formation time t as predictors. Then, they train a random forest to predict returns for each stock in the 12 months after portfolio formation to predict returns one month ahead. The top decile is bought, and the bottom decile is short, resulting in average risk-adjusted excess returns of 2 percent per month.

Krauss et al. (2017) and Fischer and Krauss (2018) compare different models under the statistical arbitrage context with the momentum strategy. Statistical arbitrage (StatArb) is an umbrella term for quantitative trading strategies generally deployed within hedge funds or proprietary trading desks, which implement strategies with the following features: "(i) trading signals are systematic, or rules-based, as opposed to driven by fundamentals, (ii) the trading book

is market-neutral, in the sense that it has zero beta with the market, and (iii) the mechanism for generating excess returns is statistical" (Avellaneda and Lee, 2010), and involve "large numbers of securities (hundreds to thousands, depending on the amount of risk capital), very short holding periods (measured in days to seconds), and substantial computational, trading, and information technology (IT) infrastructure" (Lo, 2010). In short, statistical arbitrage is the market-neutral trading strategy employing large and broadly diverse portfolios for a very short period of time. This is achieved by employing statistical methods and quantitative analysis to identify and exploit pricing inefficiencies in financial markets. To achieve this goal, they adopt a similar methodology to Takeuchi and Lee (2013). They first forecast the probability for each stock to out-/underperform the cross-sectional median in the period, making only use of information until time t. Then, they rank all stocks for each period in descending order of this probability. The top of the ranking corresponds to the most undervalued stocks that are expected to outperform the cross-sectional median. As such, they go long on the top k and short on the flop k stocks of each ranking, for a long-short portfolio consisting of 2k stocks ($k \in \{10, 50, 100, 150, 200\}$). With a lower value of k, all models perform better in terms of accuracy. In Krauss et al. (2017), the best model is an equal-weighted ensemble that consists of one deep neural network, one gradient-boosted tree, and one random forest, which shows an accuracy of around 55% and produces returns of 0.45% per day for k = 10, prior to transaction costs. In Fischer and Krauss (2018), the best model is Long-short term memory (LSTM), which shows an accuracy of 54.3% and produces daily returns of 0.46 percent for k = 10.

The methodology from Krauss et al. (2017) and Fischer & Krauss have been tested in different markets and proven to be able to generate significant returns in China and Sweden. Zhang et al. (2022) compare random forest (RF), deep neural net (DNN), XGBoost, support vector machine (SVM), and LSTM in the China stock marketplace from January 2013 to August 2017 based on Krauss et al. (2017). They find out that the random forest outperforms the others. An annual return of 124% is obtained based on the random forest, which is far beyond the HS300 index annual return (31%) during the period.

With the same methodology used in Fischer and Krauss (2018), Fjellström (2022) applies a similar methodology to OMX30, which is a capitalization-weighted index of the 30 most-traded stocks on the Nasdaq Stockholm stock exchange, from May 2002 to January 2020 using an ensemble of independent and parallel LSTMs. Compared to a randomly chosen portfolio and a portfolio containing all the stocks in the index, the LSTM-based portfolio appears to have better average daily returns and a higher cumulative return over time. The accuracy for the LSTM model is around 50%, given the complexity of financial time series data.

Based on the methodology used in Fischer & Krauss (2018), Xiang and Wang (2023) predict the intraday trading direction on CSI between the period 2005 to 2020 by utilizing TCN, LSTM, and RF, with multi-feature data. They first collect the adjusted closing and opening prices for all stocks in the CSI index from 2005 to 2020 and generate intraday returns, returns in relation to the last closing price, and returns in relation to the opening price as the input features. For RF, they convert the returns to cumulative 12-month returns and 20 daily returns based on Takeuchi and Lee (2013). For TCN and LSTM, they input 240-time steps and three features. The models are then employed to predict the probability of whether each stock will outperform the cross-sectional median of future periods. Their results demonstrate that TCN outperformed LSTM and RF using the cumulative returns. TCN shows a mean daily return of 0.852%, which is higher than LSTM (0.735%) and RF (0.705%). However, they do not analyze the model performance with accuracy. As the target variable is whether the stock is going to outperform the market median, it does not necessarily guarantee that the best-performing model will result in the highest return. We argue that the model accuracy must also be taken into account when evaluating model performance.

2.3 Trading Strategy for portfolio

While machine learning can assist in identifying promising stocks for portfolio building, a solid trading strategy is essential to fully exploit potential returns.

The trading strategy used in Fischer and Krauss (2018) is an equal-weight portfolio, which gives the same importance to each stock in a portfolio. An equal-weight portfolio provides better diversity, as it ensures each stock has an equal impact on the portfolio, which lessens the risk of concentration (Maillard et al., 2010). According to Grinold and Kahn (2000), long-short implementations demonstrate significant enhancements over long-only strategies, particularly in scenarios where the asset universe is extensive, asset volatility is minimal, and the strategy entails high active risk. Advocates of equal-weight long-short portfolios emphasize the diversification advantages offered by the short side. However, it's worth noting that while this approach can bolster portfolio stability, it may also potentially dampen profitability, particularly during favourable market conditions.

By introducing constraints on the short side, it was proven that the model could effectively increase return while balancing the increased risk compared to the equal-weighted model (Abate et al., 2022; Johnson et al., 2007). The most commonly used trading strategy proposed for generating higher profits is the 130-30 trading strategy (Lo and Patel, 2008; Frino et al., 2008; Sorensen et al., 2006; Johnson et al., 2007). A 130-30 designation implies using a ratio of 130% of starting capital allocated to long positions and accomplishing this by taking in 30% of the starting capital from shorting stocks. In other words, the 130-30 strategy shorts stocks that will underperform the market only up to 30% of the total portfolio value. Then, using these proceeds to take a long position in stocks they believe will outperform the market. For example, if the net initial investment is $100, the portfolio manager will purchase $100 of stocks expected to outperform but then short $30 worth of stocks that are expected to underperform. Then, he will use the $30 from the shorts to purchase additional stocks that are expected to outperform. The gross market exposure will thus be 160%, with 130% long and 30% short (Johnson et al., 2007). Despite the higher return, the leveraging is typically associated with higher volatility return compared to long-only portfolio (Sorensen et al., 2006).

Frino et al. (2008) test two 130/30 strategies with the top 50 stocks in the Australian Stock Exchange against the traditional buy and hold strategies as well as naive investments in market indices over an eight-year period with quarterly rebalancing. Rebalancing is the process of buying and selling portions of the portfolio in order to set the weight of each asset class back to its original state. The results of back-testing and out-of-sample simulations reveal that momentum-driven 130/30 strategies earns higher risk-adjusted returns, which is the returns from the investment that also considered the degree of risk. They evaluate the risk-adjusted returns by different metrics, including Sharpe Ratio (measures the risk to volatility trade-off), Treynor measure (measures the excess return per unit of systematic risk), Tracking error (measures the active risk of a portfolio) and Information Ratio (measures abnormal return per unit of non-systematic risk). They also mention that the transaction cost must be incorporated with the higher turnovers as this strategy will results in a higher turnover of stocks and thus higher transaction costs.

Lo and Patel (2008) exhibit the profitability of 130/30 portfolio, using different indexes over the period Jan 1996 to Sept 2007. They compare the result to different indexes, such as the S&P 500, CS/Tremont Hedge-Fund index and the Russell 1000. The result shows that the 130/30 portfolio does exhibit some performance gains over the S&P 500 and Russell 2000, even though it performs similarly compared to behaves more like traditional equity indexes than the CS/Tremont Hedge Fund Index.

Our research aims to bridge gaps in the existing literature by conducting a comprehensive analysis of state-of-the-art neural network models, including LSTM, TCN, and transformers, in the context of stock price prediction. To our knowledge, no previous study has systematically compared and examined the performance of these state-of-art models using recent years' data (2010-2017). Our goal is to assess the effectiveness of deep learning techniques in generating profitable returns from stock market data. Furthermore, existing literature lacks comparative studies on model performance across different markets using momentum strategy. Therefore, we intend to deploy our models on both the S&P 500 and DAX indices to evaluate whether their performance aligns across different marketplaces. Moreover, prior research has primarily focused on the equal-weight portfolio strategy, with no literature integrating the 130-30 trading strategy. We aim to address this gap by testing various long-short trading strategies and evaluating their profitability within the context of momentum strategy. Through this investigation, we seek to determine if employing different trading strategies can lead to higher profits in stock market trading.

3 Methodology

For the empirical application, the methodology we adopted is based on Krauss et al. (2017), Fischer and Krauss (2018) and Takeuchi and Lee (2013), which consists four steps: 1. split the data into training and testing set, 2. generate feature spaces and targets for training and making prediction, 3. train models and 4. use model and a simple ensemble to make out-of-sample predictions on the corresponding trading sets. The final output will be a binary variable $Y \in \{0, 1\}$, while 1 means the next-day return of the stock is larger than the cross-sectional median return, and 0 means otherwise. As the stocks are cut by the cross-section median, the size of the two classes has equal-sized stocks for the sake of comparability.

3.1 Data

As for the dataset, we opt for the S&P 500 index and DAX index for testing. S&P 500 index is the most important stock market index in the US that tracks the stock performance of the top 500 companies on the stock exchanges in the United States. All the data are available online and can be retrieved easily. DAX is included in Germany's stock market to see if the models perform the same across the marketplace. DAX consists of 30 major German companies for our data range, and it is a prominent benchmark for German and European stocks and an indicator of trends in Germany's economy. By testing the model on these two indices, we could test the computational feasibility in a real market. Due to the difference in the trading days, we have different data ranges for S&P 500 and DAX. For S&P 500, the data source period is from 4th Jan, 2010 to 11th Dec, 2017 for(approximately total 7 years). For DAX, the data source period is 4th Jan, 2010 until 13th Nov, 2017.

Following the methodology of Krauss et al. (2017) and Fischer & Krauss (2018), we have four steps for preprocessing the data. First, we obtain all constituent lists for the S&P 500 and DAX and from Thomson Reuters Datastream from Jan 2000 to Dec 2018 in order to eliminate survivor bias. We consolidate these lists into one binary matrix, indicating whether the stock is listed in the subsequent month or not. Then, for all the companies having ever been a constituent of the index on S&P500 and DAX during this period, we download the daily total return. Opting for daily total return indices is motivated by their incorporation of daily price changes and consideration of corporate actions such as dividends and stock splits, making them appropriate metrics for predicting the next day's stock performance. The calculated return is denoted in equation 1:

$$R_t^{m,s} = \frac{P_t^s}{P_{t-m}^s} - 1 \tag{1}$$

P_t^s: price of stock s at time t,
$R_t^{m,s}$:simple return for a stock s over m periods

Table 2 and 3 below showed the summary of the dataset.

3.2 Software and hardware

We use Python 3.9 entirely for the study. Eikon API (https://pypi.org/project/eikon/) was used to retrieve data from Refinitiv. The Numpy and Pandas library is used for data processing and feature creation. The implementation of random forest and hyperparameter tuning for the random forest is using the Scikit-Learn library (https://scikit-learn.org/). Deep learning models are built using Keras with TensorFlow backend. For hyperparameter tuning, we used the KerasTuner library (https://keras.io/keras_tuner/), which is an easy-to-use hyperparameter optimization framework that has strong integration with Keras workflows. All models are trained on a CPU (Intel Xeon, 3.00 Ghz).

Industry	No. of Stocks	Mean Return (%)	Standard derivation (%)	Skewness
Academic & Educational Services	2.3	0.0	2.3	-1.3
Basic Materials	25.7	0.1	2.0	0.5
Consumer Cyclicals	79.3	0.1	1.9	-0.2
Consumer Non-Cyclicals	46.7	0.1	1.4	2.7
Energy	41.7	0.0	2.3	0.4
Financials	65.5	0.1	1.7	-0.01
Healthcare	53.6	0.1	1.6	0.6
Industrials	58.8	0.0	1.8	-0.7
Real Estate	22.3	0.1	1.5	-0.0
Technology	73.3	0.1	1.9	1.0
Utilities	30.2	0.0	1.2	0.4
All	499.3	0.1	1.8	0.3

Table 2: Average Monthly Summary for S&P 500 from Jan, 2010 to Dec, 2017, split by TRBC economic sectors

Industry	No. of Stocks	Mean Return (%)	Standard derivation (%)	Skewness
Basic Materials	6.2	0.0	1.8	-0.1
Consumer Cyclicals	5.2	0.1	1.8	-0.3
Consumer Non-Cyclicals	1.0	0.1	1.2	0.3
Financials	5.0	0.0	2.0	0.3
Healthcare	4.0	0.1	1.2	-0.1
Industrials	3.3	0.1	1.7	-0.1
Real Estate	1.0	0.1	1.4	-0.0
Technology	3.0	0.1	1.6	0.1
Utilities	2.0	-0.0	2.0	0.0
All	30.7	0.1	1.6	0.0

Table 3: Average Monthly Summary for DAX from Jan, 2010 to Nov, 2017, split by TRBC economic sectors

3.3 Generation of training and trading sets

Before generating the features, we first split the dataset into five study periods, each comprising non-overlapping trading intervals. Following Fischer and Krauss (2018), each "study period" encompasses a training-trading set, comprising approximately 750 days (equivalent to roughly three years), followed by a trading period spanning 250 days (equivalent to roughly one year). The choice of data split is motivated by having a sufficient amount of training data and allowing non-overlapping data to be used for testing the model (trading set). Training sets are required for in-sample training of machine learning models, and trading sets are out-of-sample data that are used for evaluating performance. We then move the training-trading set forward by 250 days in a sliding window approach, resulting in 5 non-overlapping batches to loop over our entire data set from 2010-2017. As such, for S&P data, a training set consists of approximately 500*750 = 375,000 rows, while a trading set consists of approximately 500*250 = 12,000. For DAX, we have approximately 30*750 = 22,500 for the training set and 7,500 for the trading set. Figure 1 illustrated the study period and the train-test split methodology.

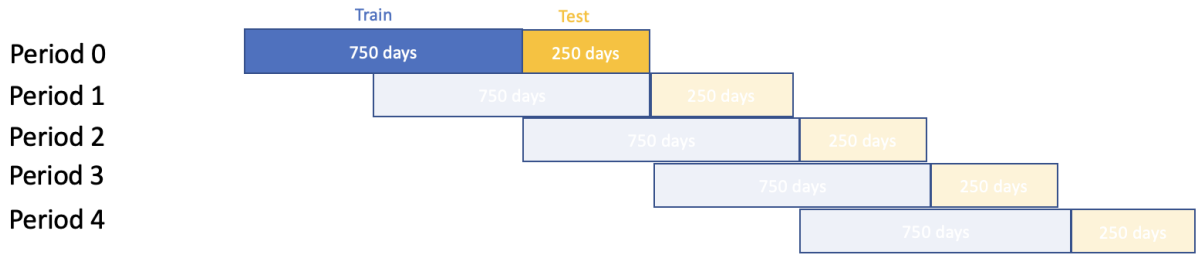

Figure 1: Study period and train-test split

3.4 Feature Generate

All models are trained on the binary classification problem of predicting whether a stock will outperform the cross-sectional median of returns on a subsequent day, based solely on the return information of 240 days, which is approximately 1 trading year of data.

The models implemented in the study can be classified mainly into two categories: one is the models with memory, namely neural network models and the models with no memory, which is the random forest. The feature generation process is different for the two types of models.

As the neural network models require time series data as input, we opt for one-day returns $R_t^{1,s}$ for each day t and each stock s. Before we generate the features space, we first standardize the returns by subtracting the mean μ_{train}^s and diving them by the standard deviation of the training period σ_{train}^s $\tilde{R}_t^{m,s} = \frac{R_t^{m,s} - \mu_{train}^s}{\sigma_{train}^s}$

With the standardized return data, we use a rolling window to retrieve the features in the form of $\{\tilde{R}_{t-239}^{1,s}, \tilde{R}_{t-238}^{1,s}, ...\tilde{R}_t^{1,s}\}$ for each stock s and each t $\geq$ 240 of the study period. If the selected data lacks a sequence length of at least 240 days, i.e. a stock is delisted from the market index between t-239 and t+1, we will exclude that particular data and proceed with the remaining set. Each data row consists of 240 variables used to predict the target. The procedure for creating input sequences with corresponding target labels is demonstrated in figure 2. For all neural network models, the input shape is 1 feature with 240 timesteps.

Feature generation is conducted separately for random forests due to the memory constraints of the model. With the previous vector, random forest treats them as individual input features rather than time series data with sequence. Since random forest cannot directly handle time series data, we create time-lagged features by aggregating returns over different intervals to capture temporal patterns in the data, making it suitable for input into the random forest model. Based on Takeuchi and Lee (2013) and Krauss et al. (2017), we use multi-period cumulative returns $R_t^{m,s}$ with lags $m \in \{\{1, 2, ...20\} \cup \{40, 60, ...240\}\}$ using the standardized return data. In other words, we first focus on the incremental daily returns of the first 20 days, which is approximately one trading month. Then, we consider the multi-period returns for the subsequent 11 months, each accounting for 20 days of return. In total, there are 31 input variables for random forest.

3.5 Targets

The binary prediction target is whether the stock is going to outperform the cross-sectional median on the next day. Therefore, for each trading day, we calculate the median for the daily return and assign class label 1 to all stocks above or equal to the cross-sectional median of returns on that day and 0 otherwise. The target class for stock s at time t, Y_t^s is then defined as:

$$Y_t^s = \begin{cases} 1, & \text{if } R^{m,s} \geq Median(R_s^m)_t, \forall s \in S(t) \\ 0, & \text{otherwise} \end{cases}$$

Figure 2: Creation of feature sequences and corresponding target labels for neural network

$R_t^{m,s}$ is the daily return for stock s on day t,
$S(t)$ is the set of stocks that is listed for trading on day t.

This results in a daily balance target and roughly equal class sizes throughout the entire dataset.

3.6 Models Training

We test and compare three state-of-art deep neural networks, namely the Long-short term memory (LSTM), Temporal convolutional network (TCN), and the transformer. Each of them are the representative of three families: recurrent neural networks (RNN), convolutional neural networks (CNN) and transformer. Random forest, feedforward neural network are used as baseline model for performance comparison.

For all neural network models, the input is 1 feature and 240 timesteps. For random forest, the input features are 31 cumulative return $R^s_{t,m}$ with $m \in \{\{1, 2, ...20\} \cup \{40, 60, ...240\}\}$, as denoted in section 3.4. Neural network models are trained with a maximum of 1000 epochs and an early stopping patient of 10.

To enhance the performance of neural network models, we adopt the following setup. First, even though the learning rate is part of the hyperparameter tuning, we start from a relatively small learning rate (1e-06). This approach ensures that the step size during each training iteration while progressing towards the optimum of the loss function, remains sufficiently small. The choice 1e-6 is motivated by its usage in Fischer and Krauss (2018). Given the large amount of training data and corresponding noise, it is important that our neural networks can learn and generalize the pattern found in the training data and not capture the noise. Therefore, we implemented three different methods to avoid overfitting. We include a dropout regularization with a 0.5 value, in which a fraction of the input units is randomly dropped at each update during training time in order to reduce the risk of overfitting and and better generalization. We also apply early stopping as a further mechanism to prevent overfitting. Early stopping tracks the model's validation set performance

throughout training, continuously assessing for signs of stopping or continuing. If such indicators emerge, suggesting potential overfitting to the training data, the training process is halted before completion. Lastly, we also include kernel weights regularization in our hyperparameter tuning. It keeps the weights of the neural network small by adding a penalizing term to the loss function.

During our training process, we conduct hyperparameter tuning to find suitable parameters, including learning rate and optimizer, for the models and maximise the model's performance. Due to the computational limitation, we tuned all neural networks with the first study period only with binary cross-entropy loss function using Bayesian Optimization search. Bayesian optimization leverages probabilistic models to efficiently explore and exploit the search space, thereby minimizing the number of evaluations needed to find the optimal solution, and considers past evaluations when choosing the hyperparameter set, resulting in faster and efficient search (Frazier, 2018). The search is implemented with KearsTuner, which provides good integration with the Keras library. For random forest, the hyperparameters are optimized separately for each study period with a random search using the classification accuracy of the respective validation fold. The search grid is summarized in table 4, and the best parameter after grid search is shown in table 5.

Model	Search Method	Parameter Grid
LSTM	Bayesian Optimization (Keras Tuner) with 30 max. trial, tuned with the first training period	Number of Memory Cell: [32, 64, 128], learning rate: [1e-06, 1e-07, 1e-08], regularizer: ['l1','l2', None], LSTM optimizer: ['sgd', 'rmsprop', 'adam', 'adamax'], Batch Size: [16,32,64,128]
TCN	Bayesian Optimization (Keras Tuner) with 30 max. trial, tuned with the first training period	Number of Filters: [32, 64, 128], Kernel Size: [2,4,8,16], learning rate: [1e-06, 1e-07, 1e-08], TCN optimizer: ['sgd', 'rmsprop', 'adam', 'adamax'], Batch Size: [16,32,64,128]
Transformer	Bayesian Optimization (Keras Tuner) with 30 max. trial, tuned with the first training period	Number of Memory Cell: [32, 64, 128], Number of transformer blocks: [2,4,8,16,32], Number of heads: [4,8,16], ff dimension of head: [2,4,8,16,32], activation for input layer: ['elu','relu', 'tanh', 'selu'] learning rate: [1e-06, 1e-07, 1e-08], Transformer optimizer: ['sgd', 'rmsprop', 'adam', 'adamax']
FNN	Bayesian Optimization (Keras Tuner) with 30 max. trial, tuned with the first training period	Unit': [32, 64, 128], learning_rate': [1e-06, 1e-07, 1e-08], optimizer': ['sgd', 'rmsprop', 'adam', 'adamax'], Batch Size: [16,32,64,128]
RF	Random Search (sklearn) with 30 max. trial, 3-fold CV, tuned with every training period	'n_estimators': [10, 100, 1000], 'max_features': ['sqrt', 'log2', None], 'max_depth': [3, 5, 7, 10, 50, None], 'criterion': ['gini', 'entropy'], 'bootstrap': [True, False]

Table 4: Parameter Grid

Model	Parameter	Best Parameter for S&P500	Best Parameter for DAX
LSTM	No. of memory cell	32	32
	Learning rate	1e-07	1e-07
	Regularizers	None	l1
	Optimizer	rmsprop	adamax
	Batch size	16	16
TCN	No. of Filters	128	128
	Kernel Size	4	4
	Learning rate	1e-05	1e-06
	Optimizer	rmsprop	adam
	Batch size	32	16
Transformer	No. of memory cell	32	128
	No. of Transformer Block	4	4
	Head Size	4	16
	ff dimension of head	8	16
	Activation	tanh	relu
	Learning Rate	1e-07	1e-08
	Optimizer	rmsprop	rmsprop
FNN	No. of memory cell	64	128
	Learning rate	1e-06	1e-06
	Optimizer	adam	adam
	Batch size	32	128
RF	number of estimator	1000,1000,1000,1000,1000	500,1000,1000,500,1000
	Maximum features	log2, log2, log2, sqrt, log2	log2, sqrt, log2, sqrt, sqrt
	Maximum Depth	None, None, None, None, None	50, None, 50, None, None
	Criterion	entropy, entropy, entropy, gini, entropy	entropy, gini, gini, entropy, gini
	Bootstrap	False, False, False, False, False	True, False, True, False, False

Table 5: Best Parameters for S&P 500 and DAX. For the Random Forest model, the hyperparameter search was conducted independently for each loop, resulting in five different sets of parameters. The first result is the best parameter for dataset 1, and so on.

Since we're dealing with a binary classification problem requiring ranking, our aim is to obtain the output in the form of the probability that the stock will outperform the cross-sectional median on the following day at t+1. The sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, is employed in all output layers of neural network models. This function maps input values to probabilities within the range of (0,1), effectively indicating the likelihood of predicting class 1. The loss function for all the neural networks is binary cross-entropy, which is a common setup for the binary classification problem.

3.6.1 Novel Models

Three state-of-the-art models have been implemented for comparison in our study. These models were selected based on their distinct neural network architectures, each of which is well-suited for time series prediction. LSTM (Long Short-Term Memory) is a representative of recurrent neural networks (RNNs), that has its memory cells and gating mechanisms for time series data. TCN (Temporal Convolutional Network) stands as a representative of convolutional neural networks (CNNs) that utilize dilated convolutions to efficiently capture long-range dependencies in time series data. Transformer

is a type of neural network model based on attention mechanisms and able to capture global dependencies in sequences, making it particularly suitable for tasks such as language translation and sequence prediction.

LSTM The movement of stock depends not only on the short-term features but also on long-term features (Krauss et al., 2017). The traditional neuron network assumes that the input and output variables are independent of each other and not able to learn the dependencies between features. LSTM, which is based on RNN, is not only able to learn the dependencies between variables but also overcomes the vanishing gradient descent problem using long-term and short-term memory to learn long-term dependency. It is commonly used in time series prediction. Therefore, it would be interesting to compare LSTM performance to TCN.

Following Fischer and Krauss (2018), we apply a 1-layer LSTM model with a training duration of 1000 epochs and an early stopping patience of 10. The specified topology of our LSTM network is as follows:

- Input layer with 1 feature (daily return) and 240 timesteps
- LSTM layer. The number of memory cells, learning rate, regularizer, and optimizer are part of the hyperparameter tuning.
- Dropout layer with 0.5 dropout value.
- Output layer (dense layer) with 1 neurons and sigmoid as activation function

The hyperparameter tuning for LSTM involves the number of memory cells, learning rate, regularizer, optimizer and batch size.

- Number of Memory Cells: defines the complexity and capacity of the LSTM model. Tuning this parameter allows us to find the optimal balance between model complexity and generalization capability.
- Learning Rate: governs how quickly the network updates its parameters during training. A higher learning rate accelerates the convergence of the optimization process, but it may also result in overshooting the optimal solution or getting stuck in local minima. Conversely, a lower learning rate ensures smoother optimization but may lead to slower convergence or getting trapped in saddle points. The minimum amount of learning rate is le-6, which is motivated by the learning rate used in Krauss et al. (2017).
- Regularizer: prevents overfitting by penalizing the model's complexity, especially when we have a large number of epochs (up to 1000 epochs) for training. By tuning the regularizer parameter, we control the magnitude of the penalty imposed on the model's weights to avoid overfitting and allow the model to learn more robust and generalizable patterns.
- Optimizer: updates the model's parameters based on the computed gradients during training. By tuning the optimizer parameter, we aim to find the most suitable optimization algorithm that accelerates convergence and improves the model's performance.
- Batch Size: determines the number of samples processed by the model in each training iteration. Tuning the batch size affects the stability of the training process, the memory requirements, and the computational efficiency. A smaller batch size provides more frequent updates to the model's parameters and introduces more stochasticity, while a larger batch size offers smoother optimization but requires more memory. We offer different batch sizes for the model to select, starting from 16 and up to 128.

Temporal convolutional network A Temporal Convolutional Network (TCN) is a specialized deep learning architecture for sequential data analysis, ideal for tasks like time series forecasting and natural language processing. TCNs employ convolutional layers with dilated convolutions to efficiently capture temporal patterns over long sequences while maintaining computational efficiency (Bai et al., 2018). They often utilize residual connections for easier training and better performance. TCNs are known for their ability to handle long-range dependencies, parallel processing,

and state-of-the-art performance in various sequential data tasks. Compared to traditional RNN, TCN offers a higher parallelism and requires less memory consumption.

We apply the TCN model with the training duration of 1000 epochs and an early stopping patience of 10. The specified topology of our TCN network is as follows:

- Input layer with 1 feature (daily return) and 240 timesteps
- TCN layer. The number of memory cells, kernel size learning rate, regularizer, and optimizer are part of the hyperparameter tuning.
- Dropout layer with 0.5 dropout value.
- Output layer (dense layer) with 1 neurons and sigmoid as activation function

The hyperparameter tuning for TCN involves the number of filters, learning rate, regularizer, optimizer and batch size.

- Number of Filters: determines how many different features or patterns the layer can detect in the input data. Each filter learns to detect a specific feature or pattern by convolving across the input data using its weights. The number of filters is similar to the units for LSTM, so the input grid is the same as the number of memory cell used in LSTM.
- Kernel Size: controls the spatial area/volume considered in the convolutional ops. As suggested by the Keras library, good values are usually between 2 and 8.

Transformers Transformer (TF) use attention mechanisms to weigh the importance of different elements in the input sequence, enabling them to capture complex dependencies across the entire sequence length in parallel. It can learn about the long-term dependencies with the self-attention mechanisms and selectively attend to different parts of the input sequence. Transformers consist of encoder and decoder layers, each composed of multi-head self-attention mechanisms and feedforward neural networks. Additionally, Transformers may include positional encoding to provide information about the position of tokens in the sequence. In the context of financial time series data, Transformers are expected to put emphasis on and extract crucial historical features from the noisy historical data, thereby enhancing their ability to discern meaningful patterns and trends.

We implemented the transformer architecture from the paper "Attention is all you Need" (Vaswani et al., 2017), with the training duration of 1000 epochs and early stopping patience of 10. The specified topology of our transformer network is as follows:

- Input layer with 1 feature (daily return) and 240 timesteps
- Transformer layer with dropout value 0.5. The number of memory cells, learning rate, regularizes, and optimizer is part of the hyperparameter tuning.
- Output layer (dense layer) with 1 neurons and sigmoid as activation function

The hyperparameter tuning for the transformer involves the number of cells, number of transformer blocks, number of heads, ff dimension of the head, activation layer, learning rate and optimizer.

- Number of Cells: determines the dimensionality of the input and output vectors at each position in the Transformer model. Tuning this parameter controls the complexity and capacity of the model.
- Number of Transformer Blocks: The Transformer architecture consists of multiple stacked Transformer blocks, each containing self-attention and feed-forward neural network layers. Tuning the number of blocks adjusts the depth and capacity of the model.
- Number of Heads: Self-attention mechanisms in Transformers are composed of multiple attention heads, each capturing different aspects of the input sequence. Tuning the number of heads controls the granularity and diversity of attention patterns learned by the model.

- FF Dimension of Head: determines the dimensionality of the hidden layer in the feed-forward networks within each Transformer block. Tuning it adjusts the capacity and complexity of the feed-forward networks.

3.6.2 Benchmark Models

Other than the novel models, we implemented random forest and feedforward neural network as baseline models for benchmarking performance.

Random Forest Random forest is an example of an ensembling method, which consists of a large number of individual decision trees. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The algorithm's strength lies in its ability to handle complex datasets and mitigate overfitting, making it a valuable tool for various predictive tasks in machine learning.

We use a random forest as a baseline model for two reasons. First, random forest is a machine learning model that can be implemented easily with relatively low computational requirements and usually delivers good results. Second, random forest achieves the best accuracy in Krauss et al. (2017) compared to deep neural networks, gradient boost trees, and the ensembling models of these methods from 1992 to 2015.

We implement the random forest through the Scikit-Learn library with the default setting, then tune the number of estimators, maximum features, whether to use Bootstrap and whether to use gini or entropy to measure the quality of a split. With Scikit-Learn RandomizedSearch, we apply 3-fold cross-validation to find out the best-performed parameter using random search. The hyperparameter tuning for random forest involves the number of estimators, maximum features, maximum depth, criterion and bootstrap.

- Number of Estimators: the number of decision trees to be used in the Random Forest ensemble.
- Maximum Features: the number of features selected at random and without replacement at split. A higher number of features will generally lead to a more accurate model, but it can also lead to overfitting.
- Maximum Depth: controls the maximum number of levels in each tree, which can help prevent overfitting by limiting the complexity of individual trees. However, setting it too low may result in underfitting.
- Criterion: measures the quality of a split in the decision trees.
- Bootstrap: controls whether bootstrap samples are used when building decision trees. Setting it to True enables bootstrap sampling, which introduces randomness into the training process and helps improve model robustness.

Feedforward Network A feedforward neural network (FNN) is an artificial neural network with a circular network of nodes. Its information flow is unidirectional, meaning that data in the model travels exclusively in one direction—from the input nodes, through any hidden node, and to the output nodes—without forming any loops or cycles. Feedforward networks are typically trained using the backpropagation method. In contrast to other neural network architectures utilized in our paper, the FNN is comparatively simpler, typically comprising fewer layers of nodes. Consequently, the training time and computational resources needed for FNNs are reduced.

We apply the standard multi-layer feed-forward neural network with one hidden layer as a baseline model to see how well a simple neural network model can do. The specified topology of our FNN network is as follows:

- Input layer with 1 feature (daily return) and 240 timesteps
- Dense layer. The number of memory cells, learning rate, regularizes, and optimizer is part of the hyperparameter tuning.
- Dropout layer with 0.5 dropout value.
- Output layer (dense layer) with 1 neurons and sigmoid as activation function

3.7 Forecasting and portfolio formation

At the end of each trading day t, we predict the probability of outperforming the cross-sectional median on day t+1, denoted by P^s_{t+1} independently for each stock using the previous 240 days of daily return for all models. To obtain the final class prediction per model type for each stock per day, the stocks for each day are sorted by the probability of being class 1. The top of the ranking corresponds to the most undervalued stocks that are expected to outperform the cross-sectional median in t+1, and vice versa.

To create the portfolio, we go long for the top k stocks and short the flop k stocks for each ranking for a long-short portfolio consisting of 2k stocks. For S&P500, we choose a portfolio size where $k \in \{10, 50, 100, 150, 200\}$. For DAX, as the maximum number of stocks is 30, we choose a portfolio size where $k \in \{2, 4, 5, 8, 10, 15\}$.

To enhance profitability, we aim to explore various trading strategies. We construct our portfolio by incorporating two distinct trading approaches as follows:

- market neutral strategy: we long the top k stocks and short the bottom k stocks with equal weights.
- 130-30 strategy: short 30% of the bottom k stocks, use the money from short selling to long the top k stocks.

3.8 Performance analysis

The performance analysis for the models is mainly divided into two parts. First, we look at the accuracy of the models with different portfolio sizes. Secondly, we look into the portfolio's financial performance with returns and risk-return characteristics using the Sharpe Ratio.

Based on our long-short trading strategy and the portfolio selection rule described above, we first assess the accuracy for various values of k. Each portfolio position is initiated at the end of day t, based on the market closing prices, and closed at the end of day t+1 after a holding period of one full day. During each open-close cycle, the portfolio incurs a transaction cost of 0.5% per share per half-turn, following the approach outlined in Krauss et al. (2017). Given the daily transactions and high turnover of the trading strategy, transaction costs must be taken into account. After each position is closed at the end of the holding period, the resulting cash position is utilized to fund the subsequent day's trades. For each model, we maintain a long-short portfolio comprising 2k stocks at any given time, adjusting its composition once per day while incurring transaction costs for every trade. The financial performance of this daily trading strategy is determined using the net asset value of investments, incorporating both stock returns and incurred trading costs.

3.9 Difference with previous studies

Despite our methodology being mainly based on Krauss et al. (2017) and Fischer & Krauss (2018), there are several differences between our methodology and those outlined in these two papers. Firstly, we extend our analysis to include the German stock market index DAX, allowing for comparison across different stock markets. Secondly, we opt for 2010 to 2017 for our dataset selection period, which differs from Fischer & Krauss (2018), spanning from December 1989 to September 2015. We aim to assess our models using a relatively new dataset while still maintaining an overlapping period for comparison and validation purposes. Due to the computational constraints, we opt for a shorter data period of 7 years (5 data periods) instead of the 25 years (23 data periods) utilized by Krauss et al (2017). Lastly, Fischer & Krauss (2018) implement an equal-weight trading strategy in their portfolio, where equal amounts of capital are allocated to both long and short positions. In line with their study, we aim to test an alternative trading strategy, specifically the 130-30 strategy, to potentially maximize profits. The 130-30 trading strategy involves allocating 130% of the portfolio's value to long positions while shorting 30%. By incorporating this strategy into our analysis, we seek to compare its effectiveness against the equal-weight strategy employed by Fischer & Krauss (2018), providing valuable insights into the optimal allocation of resources for maximizing returns in financial markets.

4 Result and Discussion

4.1 Model Evaluation

In this section, we present the results of different models in terms of model evaluation using accuracy and financial performance using a long-short trading strategy.

4.1.1 Model Accuracy

The employed model's ability to accurately predict whether a stock is going to outperform the cross-sectional median is the basis for forming a profitable long-short portfolio. Therefore, we first evaluate the models in terms of their accuracy, which is calculated as $\frac{TP+TN}{TP+FP+FN+TN}$. The accuracy of all models remains relatively consistent, slightly exceeding 50%. The complete summary of prediction accuracy for long-short portfolio per model with different portfolio sizes is shown in Table 6 for S&P 500 and Table 7 for DAX.

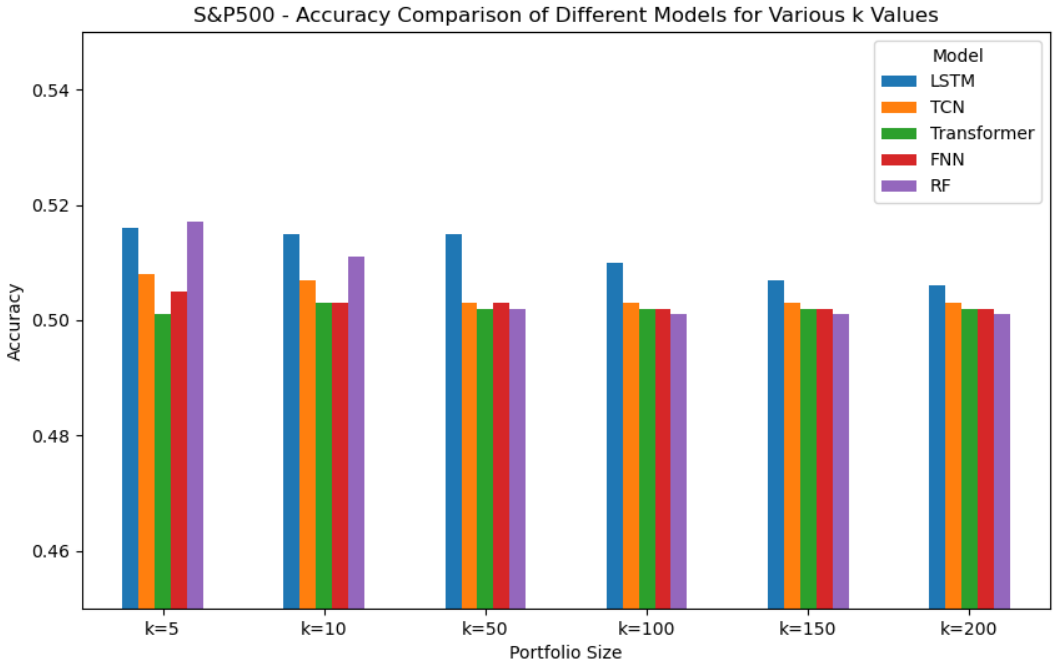

Figure 3: Prediction accuracy for different portfolio sizes for S&P500

Model	k=5	k=10	k=50	k=100	k=150	k=200
LSTM	0.516	0.515	0.514	0.51	0.507	0.506
TCN	0.508	0.507	0.503	0.503	0.503	0.503
TF	0.501	0.503	0.502	0.502	0.502	0.502
FNN	0.505	0.503	0.503	0.502	0.502	0.502
RF	0.517	0.511	0.502	0.501	0.501	0.501

Table 6: Prediction accuracy for S&P500 with different portfolio size

For the S&P 500 index, LSTM consistently outperforms other models across various portfolio sizes, ranging from $k = 10$ to $k = 200$, except for the $k = 5$ scenario where random forest performs the best with an accuracy of 51.7%.

However, the Transformer model exhibits the lowest accuracy among all models, ranging from 50.1% at $k = 5$ to 50.3% at $k = 10$. TCN's performance falls in the middle, with accuracies ranging from 50.8% at $k = 5$ to 50.3% for larger values of k ($k \in \{50, 100, 150, 200\}$). FNN also struggles to predict the stock performance accurately for the S&P 500, with the accuracy ranging from 50.5% a k=5 to 50.2% at ($k \in \{100, 150, 200\}$). Overall, there is a trend of decreasing accuracy with larger values of k, indicating that as the portfolio size increases, the models struggle to maintain their predictive performance.

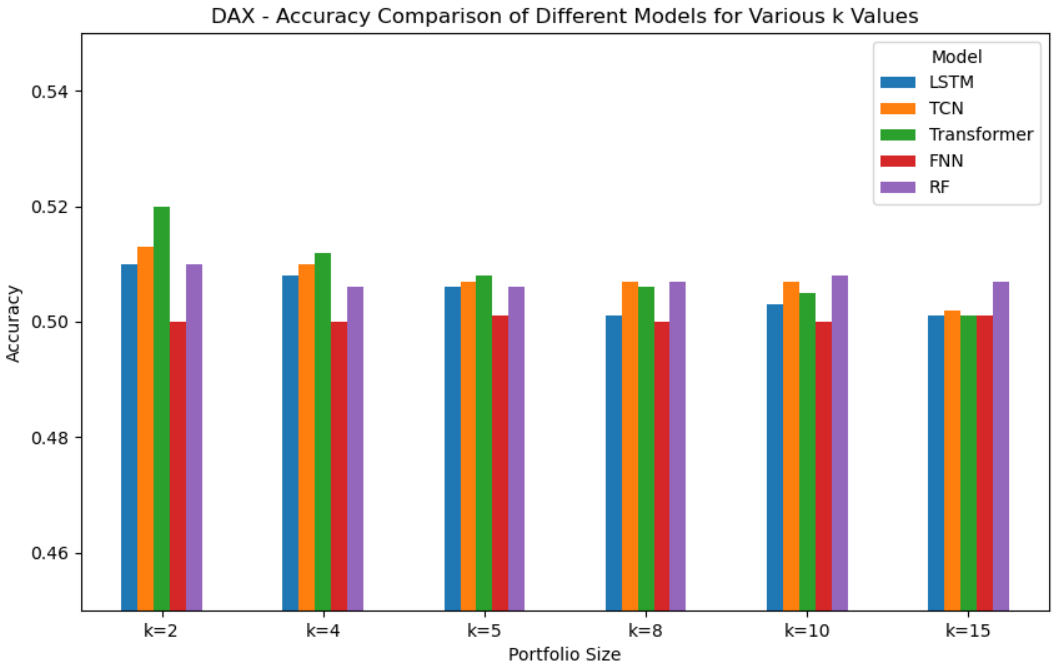

Figure 4: Prediction accuracy for different portfolio sizes for DAX

Model	k=2	k=4	k=5	k=8	k=10	k=15
LSTM	0.51	0.508	0.506	0.501	0.503	0.501
TCN	0.513	0.51	0.507	0.507	0.507	0.502
TF	0.52	0.512	0.508	0.506	0.505	0.501
FNN	0.5	0.5	0.501	0.5	0.5	0.501
RF	0.51	0.506	0.506	0.507	0.508	0.507

Table 7: Prediction accuracy for DAX with different portfolio size

The result is in line with previous research findings spanning the 2010-2015 period (Fischer and Krauss, 2018). In their study, the 1993-2000 period reported higher accuracy rates, ranging from 55% to 60%. Notably, the accuracy drops to approximately 51% to 52% for the 2010-2015 timeframe. In the meantime, the random forest also lost its edge and resulted in a negative average daily return. They attribute this change to the increased volatility of the finance market. Moreover, financial time series data is inherently noisy, making it challenging to detect capital market anomalies. The substantial profits observed from 1993 to 2000 can be attributed to the limited awareness and feasibility of LSTM models among market professionals during that period.

For the DAX index, the performance of different models varies depending on the portfolio size. Specifically, the transformer exhibits superior performance for smaller portfolio sizes, with accuracy rates ranging from 51% to 52% for $k \in \{2, 4, 5\}$. In contrast, the Random Forest model outperforms the others for larger portfolio sizes, maintaining an accuracy level of approximately 51% for $k \in \{8, 10, 15\}$. Conversely, the Feedforward Neural Network (FNN) consistently demonstrates lower accuracy across all portfolio sizes, ranging from 50% to 50.1%. The Temporal Convolutional Network (TCN) displays moderate performance, with accuracy ranging from 51.3% for $k = 2$ to 50.2% for $k = 15$. LSTM's performance, although relatively lower than TCN, still exhibits competitive accuracy rates, ranging from 51% at $k = 2$ to 50.1% for $k \in \{8, 15\}$. Similar to the observations for the S&P 500 index, there is a general trend of decreasing accuracy with an increase in portfolio size.

Following Fischer and Krauss (2018), we provide a statistical estimate for the probability of models having randomly achieved this accuracy. Unlike the previous studies (Fischer and Krauss, 2018; Krauss et al., 2017), we observe that there is no single portfolio size at which all models demonstrate the highest accuracy, even though as the portfolio size increases, the accuracy of machine learning models tends to decline in general. Given the overall model accuracy performance and considering the need to diversify risk while maximizing model performance, we opt for a k=10 portfolio for both S&P 500 and DAX. For k =10, we consider a total of 250 (trading day a year) * 7 (duration of the dataset) * 20 (portfolio size) = 35,000 top and flop stocks. For an accuracy =50%, we model the number of correctly classified stocks under the binomial distribution where $X_k \sim \mathbf{B}(n = 35,000, p = 0.5)$, and calculate the probability of a random classifier performing the accuracy by chance alone. We import the binomial package from the Scipy Stats library and calculate the corresponding p-value, and all models have a significant p-value of 0.05.

Based on the results obtained, it is challenging to conclude which model performs the best for the finance prediction task. Across different markets and varied portfolio sizes, the performance of the assessed models exhibits considerable variability, with the top-performing model achieving an accuracy ranging from 51% to 52%. Notably, as the portfolio size increases, the accuracy of machine learning models tends to decline. This variability suggests that each model may perform differently under specific conditions or contexts within the financial market. As a recommendation for investors, a comprehensive examination of diverse models under various market conditions and portfolio sizes is advised to discern which ones offer greater reliability within a particular framework. Such thorough testing can provide valuable insights into model performance and aid in selecting the most suitable approach for financial prediction tasks.

4.2 Financial Performance

The accuracy alone, however, does not provide a complete picture of the model's performance since investors actually care more about returns in their objective function. Investors prioritize returns in their objective function, emphasizing the need to generate high profits from investment strategies. Therefore, in addition to evaluating prediction accuracy, our analysis will also focus on assessing the profitability of each model and trading strategy. By considering both accuracy and profitability, we aim to provide a more holistic evaluation of the models' effectiveness in real-world investment scenarios. Considering the overall model accuracy performance and the need to diversify risk, all the financial performance analyses are based on k=10 for both S&P 500 and DAX, a diverse portfolio with 10 long and 10 short positions.

Since our target variable indicates whether a stock will outperform the cross-sectional market median, having a better performance model doesn't guarantee the highest profit. To provide a comprehensive evaluation and benchmark the model prediction against the market performance, we compare our model results with those of two random market portfolios.

- Random Portfolio 1: Each day, we randomly select the top 10 stocks to buy and the bottom 10 stocks to sell short. We then execute these buy and sell transactions daily, adjusting our portfolio accordingly.

- Random Portfolio 2: At the beginning of the period, we randomly select the top 10 stocks to buy and the bottom 10 stocks to sell short. We maintain these positions without making any further buy or sell decisions until the end of the period.

4.2.1 Daily Returns

We first evaluate the financial performance with the daily return. For an equal-weight strategy, an equal amount of capital is allocated to both the long and short portfolios. For the 130-30 weight strategy, capital allocation involves investing 130% of the portfolio's value in long positions while simultaneously shorting 30%, which utilized the potential gains from long positions and hedges against the downside with the short. This means that we generate the return of the stock by 130% of the daily return for long positions and 30% of the daily return for short positions. Under the equal weight strategy, all models exhibit positive cumulative returns by the end of the period, despite fluctuations in the interim. Conversely, employing the 130-30 strategy yields significantly higher returns compared to both random market portfolios.

We integrate transaction costs into our analysis by implementing a daily rebalancing strategy, which incurs a transaction cost of 0.5% for each half-way transaction, following the methodology applied in Fischer and Krauss (2018). This implies that adjustments to the portfolio holdings are made daily, considering the transaction expenses involved. Transaction costs are incurred for every buy and sell transaction. By accounting for transaction costs in this manner, we aim to provide a more accurate reflection of the real-world challenges and expenses faced by investors in executing trading strategies.

S&P 500 Daily Return Table 8 presents the performance characteristics of the equal-weight portfolio and 130-30 strategy applied to the S&P 500 dataset spanning from January 2012 to December 2017.

For an equal-weight portfolio, among the various models considered, the TCN model demonstrates the most promising performance with 2.7% after transaction costs. Notably, it maintains a moderate level of volatility with a standard deviation of 1.040, indicating relatively stable performance. Furthermore, the TCN model exhibits strong risk-adjusted performance, as evidenced by a Sharpe ratio of 3.1 after transaction costs, positioning it as one of the top performers in terms of risk-adjusted returns. LSTM follows closely with 2.5% return after transaction costs, but with the highest standard deviation of 1.471 among all models. The third best model is random forest with a mean return of 2.1% after transaction costs, while maintaining the lowest volatility level with a standard deviation of 0.843. The transformer model ranks fourth in terms of mean return, yielding 0.6% after transaction costs, and demonstrating the lowest volatility among all models with a standard deviation of 0.858. Lastly, the FNN model presents the lowest mean return of 0.4% before transaction costs and -0.5% after transaction costs, accompanied by a relatively high standard deviation of 1.381, indicating higher volatility compared to other models. It is also the only model with a negative daily return and performs worse compared to a random portfolio after considering transaction cost.

In 130-30 trading, aligning with the previous studies (Abate et al., 2022; Johnson et al., 2007), the mean daily return for all models is notably higher than that of the equal-weight return, but the corresponding standard deviation also increases. The rational behind is the 130-30 trading strategy allocates more capital on the long side, which reduces the benefit that the short side on eliminating the risk, thereby the portfolio is more volatile. Under the equal-weight portfolio, the standard deviation for all models falls within the range of 0.6 to 0.7. In contrast, for the 130-30 trading portfolio, the standard deviation ranges between 1.1 and 1.4.

After factoring in transaction costs, all models in the 130-30 trading strategy exhibit significantly higher daily returns compared to the random market portfolio. Among the models analyzed, the TCN model outperforms the others, achieving a daily return of 9.9% after accounting for transaction costs, which is consistent with the performance of the equal-weight portfolio. Importantly, it maintains a moderate level of volatility with a standard deviation of 1.215, indicating relatively stable performance. Following closely, the LSTM model achieves a mean return of 8.1% after

transaction costs, demonstrating the second-best performance. The FNN model ranks third, achieving a daily return of 7.8% after transaction costs, with the highest standard deviation of 1.761. Both the transformer model and the random forest model yield a daily return of 6%, which are both the lowest among the models considered but still notably higher compared to a random portfolio.

DAX Daily Return Table 9 presents the performance characteristics of the equal-weight portfolio and 130-30 strategy applied to the DAX dataset spanning from January 2012 to November 2017.

For the equal weight portfolio, FNN achieves the highest daily return of 1% after transaction cost. Following closely behind are the LSTM and random forest models, both exhibiting an average daily return of 0.9% after transaction costs. Subsequently, the TCN model yields a daily return of 0.3% after transaction costs, while the transformer model presents the lowest prediction accuracy, with a daily return of 0.1% after transaction costs. All models show a higher mean daily return compared to the two market random portfolios.

With the 130-30 trading strategy, aligning with the S&P 500 portfolio performance characteristics, we see also a significantly higher daily return and a higher standard deviation compared to an equal-weight portfolio. The transformer achieves the highest daily return of 6.5%, with a moderate standard deviation of 1.274. The second best model is the random forest, which achieves the mean return of 6.4% with the lowest standard deviation (1.166) among all model predictions. The TCN model ranks third in terms of performance, yielding a daily average return of 6.1% after transaction costs, while the FNN and LSTM models follow closely behind, with daily average returns of 5.7% and 5.6%, respectively.

4.2.2 Risk-return characteristics

To evaluate the portfolio's volatility and risk relative to its return, we utilize the annualized Sharpe ratio, denoted as $\frac{(R_x - R_f)}{\text{Standard Deviation}}$. The Sharpe ratio measures the excess return per year compared to the risk-free rate, divided by the standard deviation of the portfolio returns, thus quantifying the excess return per unit of risk (Sharpe, 1998). There is no standard definition for a 'good' Sharpe Ratio. Usually, any Sharpe Ratio greater than 1.0 is considered an acceptable good, which offers excess returns relative to volatility, while a ratio under 1.0 is considered sub-optimal (Investopedia, nd). In Fischer and Krauss (2018), the annualized Sharpe Ratio is 5.83 for LSTM before transaction cost and 2.34 for LSTM after transaction cost.

For the United States, we consider the average interest rate for U.S. 10-year Treasury bonds from 2012 to 2017 as the risk-free rate. The monthly interest rate is recorded at 2.17% (source: St. Louis Fed, as shown in figure 5). To annualise this rate, we use the compounded annual rate formula: $(1 + r)^m - 1$, where r represents the monthly interest rate and m denotes the period (12 months). This yields an annual rate of 29.4%. Similarly, for Germany, we utilize the interest rate for 10-year German government bonds from 2012 to 2017, approximately 0.8% monthly, translating to around 10% per annum (source: St. Louis Fed, shown in figure 5).

S&P 500 Sharpe Ratio For the equal-weight trading strategy, the Random Forest (RF) model demonstrates the highest Sharpe Ratio among all models at 4.989 after accounting for transaction costs. These figures indicate superior risk-adjusted returns, highlighting the efficacy of the RF model in managing risk relative to its returns. Following closely, the TCN model ranks second with a Sharpe Ratio of 3.135. However, TCN shows the highest annual return, its higher standard deviation results in a lower return-to-risk ratio. The Transformer (TF) model takes third place with a Sharpe Ratio of 2.3 per annum. Having the highest standard deviation (1.37) among all models, LSTM's Sharpe Ratio ranks fourth with 2.02, even though its annual return is the second-highest.

Conversely, the Feedforward Neural Network (FNN) exhibits negative Sharpe Ratios (-0.637) after transaction costs, suggesting that the predictions from FNN do not yield good risk-adjusted returns. FNN exhibits a high level of fluctuation during the period Jan 2016 to Jun 2016 (Appendix B). This instability in return predictions results in negative Sharpe Ratios after transaction costs.

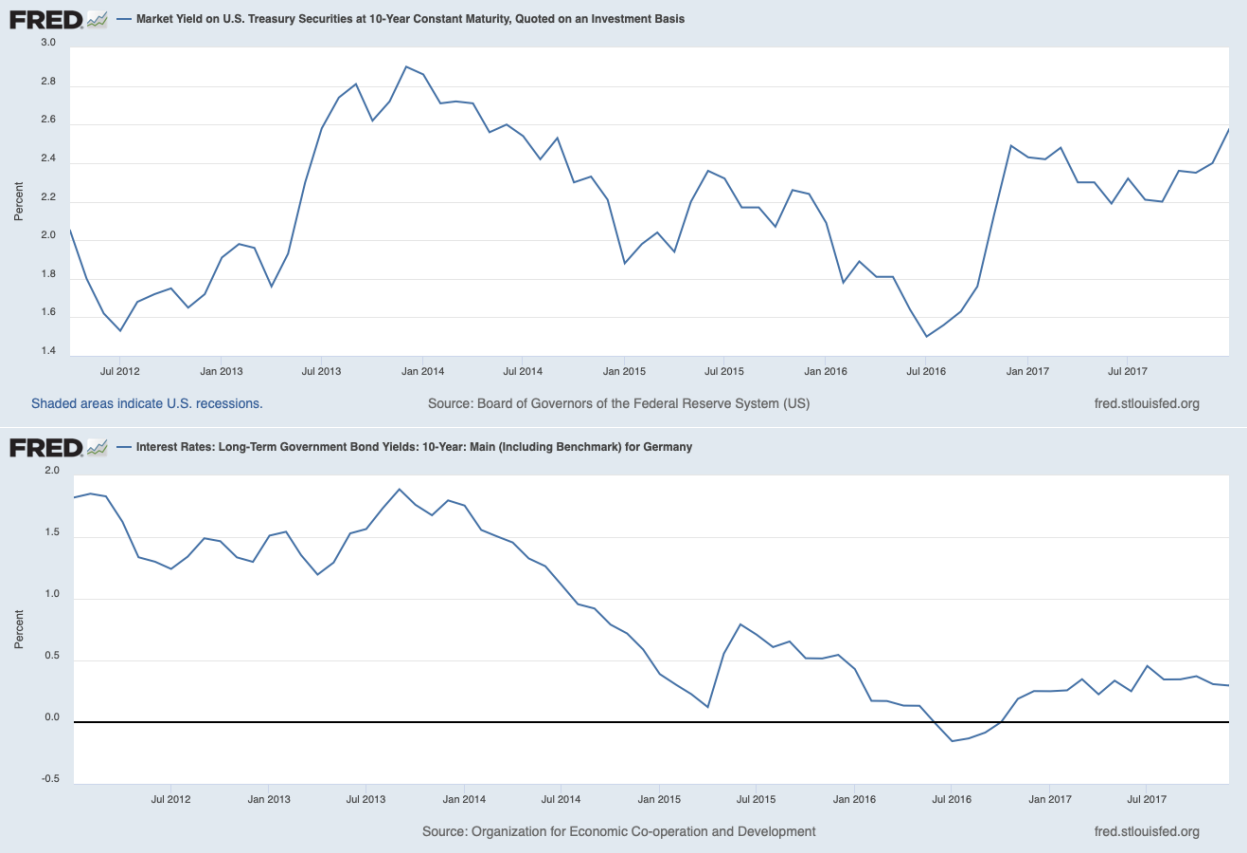

Figure 5: Upper: Market Yield on U.S. treasury securities at 10-Year Constant Maturity, quoted on an investment basis. Bottom: Interest Rate for 10-year Government Bond Yields for Germany. Source from St. Louis Fed.

For the 130-30 trading strategy, all models demonstrate notable positive returns and exhibit high Sharpe ratios despite experiencing a doubling in standard deviation per annual compared to the equal weight trading approach. Among the models, TCN stands out with the highest Sharpe ratio of 15.598 per annum after transaction costs. Following closely are RF and TF, with Sharpe ratios of 13.371 and 13.285 per annum, respectively. FNN ranks fourth with a Sharpe ratio of 11.249 per annum. Even though LSTM ranks as the least performing model with a Sharpe ratio of 5.387 per annum, its performance still surpasses that of the market random portfolios in terms of the Sharpe ratio.

Comparing to the random market portfolio, all models exhibit lower standard deviations and less extreme minimum and maximum returns for both equal weight trading and 130-30 trading strategy, suggesting reduced volatility in their performance. This enhanced stability underscores the effectiveness of the models in managing risk and delivering more consistent performance outcomes compared to the choosing stock randomly.

DAX Sharpe Ratio For equal-weight trading, LSTM stands out as the top performer with a Sharpe Ratio of 3.686, followed by RF with 2.672. FNN takes the third position with a notable Sharpe Ratio of 2.529. In comparison, TCN and TF exhibit lower Sharpe ratios of 3.686 and 0.631, respectively. Importantly, all Sharpe Ratios surpass those of the market portfolio.

For 130-30 trading, unlike the random market portfolio, which all exhibit negative Sharpe Ratios, our models achieve positive and significant Sharpe Ratios after transaction costs. LSTM once again performs the best among all models, achieving a notably high Sharpe Ratio of 12.159 per annum after transaction costs. Similarly, RF secures the second

position with a Sharpe Ratio of 11.807 after transaction costs, followed closely by TF with a Sharpe Ratio of 11.515. TCN and FNN demonstrate comparable performance, each with a Sharpe Ratio of 10.708 and 10.175, respectively.

From a finance perspective, while it's challenging to definitively determine which model performs the best, it's evident that machine learning combined with a momentum strategy can enhance profitability and maintain a positive risk-adjusted return compared to randomly selecting stocks in the market, indicated by the high Sharpe Ratio compared to the market random portfolios. Specifically, these models demonstrate lower standard deviations, indicating a narrower range of returns around the mean, and less extreme minimum and maximum returns. This observation suggests that the investment strategies employed by these models lead to more stable and predictable outcomes compared to randomly constructed portfolios. This stability and predictability are crucial for investors seeking consistent returns and risk management in their investment decisions. Furthermore, the higher Sharpe Ratios exhibited by these models, particularly in the 130-30 trading strategy, underscore their ability to generate superior risk-adjusted returns, reinforcing their appeal to investors seeking optimal risk-return profiles.

S&P500 Return Performance Characteristics												
	Before transaction cost					**After transaction cost**					**Market**	
Model	**LSTM**	**TCN**	**TF**	**FNN**	**RF**	**LSTM**	**TCN**	**TF**	**FNN**	**RF**	**RND 1**	**RND 2**
Equal Weight												
Mean Return	0.031	0.036	0.015	0.004	0.030	0.025	0.027	0.006	-0.005	0.021	-0.003	-0.039
Standard Dev	1.471	1.040	0.858	1.381	0.843	1.471	1.040	0.858	1.381	0.843	1.7	1.632
Minimum	-9.05	-7.078	-7.547	-9.287	-4.253	-9.06	-7.086	-7.557	-9.296	-4.263	-41.111	-42.223
Quartile 1	-0.681	-0.530	-0.450	-0.695	-0.421	-0.687	-0.538	-0.459	-0.704	-0.430	-0.747	-0.069
Median	0.013	0.044	0.029	0.001	-0.001	0.008	0.035	0.019	-0.005	-0.009	0.000	-0.069
Quartile 3	0.76	0.579	0.463	0.681	0.480	0.755	0.571	0.453	0.673	0.472	0.757	0.733
Maximum	8.258	4.718	5.318	6.712	4.567	8.249	4.709	5.309	6.705	4.559	38.432	22.839
Return p.a	6.539	7.515	3.183	0.761	6.263	5.258	5.696	1.176	-1.024	4.323	-0.015	-0.025
Excess Return p.a	6.246	7.221	2.889	0.468	5.969	4.965	5.402	0.882	-1.318	4.029	-0.309	0.319
Standard Dev. p.a.	1.370	0.915	0.832	1.220	0.798	1.37	0.914	0.832	1.219	0.798	1.608	1.605
Sharpe Ratio p.a.	2.975	5.057	4.786	0.817	7.399	2.02	3.135	2.300	-0.637	4.989	-0.2	0.2
130-30 Trading												
Mean Return	0.087	0.106	0.068	0.085	0.066	0.081	0.099	0.060	0.078	0.058	0.019	-0.013
Standard Dev	1.705	1.215	1.064	1.761	1.010	1.705	1.215	1.064	1.761	1.010	1.594	1.535
Minimum	-11.817	-5.174	-6.805	-9.110	-4.493	-11.825	-5.181	-6.813	-9.116	-4.501	-24.434	-21.63
Quartile 1	-0.702	-0.579	-0.510	-0.825	-0.442	-0.705	-0.586	-0.517	-0.833	-0.450	-0.405	-0.581
Median	0.113	0.106	0.112	0.115	0.092	0.107	0.099	0.104	0.107	0.084	0.0	-0.031
Quartile 3	0.931	0.759	0.714	1.005	0.619	0.927	0.754	0.706	0.997	0.611	0.452	29.691
Maximum	9.232	7.347	4.133	11.344	6.908	9.226	7.340	4.125	11.339	6.901	49.962	0.445
Return p.a	18.104	22.151	14.116	17.687	13.648	17.037	20.687	12.506	16.257	12.094	0.014	0.012
Excess Return p.a.	17.819	21.858	13.822	17.393	13.354	16.743	20.393	12.212	15.963	11.800	-0.279	-0.282
Standard Dev. p.a.	1.765	1.181	1.060	1.672	0.999	1.765	1.181	1.060	1.672	0.999	1.520	1.456
Sharpe Ratio p.a.	10.388	16.893	14.852	12.167	15.009	9.698	15.598	13.285	11.249	13.371	-0.188	-0.195

Table 8: Performance characteristics of the k=10 equal-weight portfolio (top) and 130-30 portfolio (bottom) for S&P500 from Jan 2012 to Dec 2017. The first part represents the daily return statistic, while the second part represents the per annual return statistic.

5 Conclusion

The study underscores the potential of machine learning in stock price prediction, as evidenced by the models' significant accuracy outperformance compared to random chance alone. Moreover, machine learning, when applied with a momentum strategy, yields considerable returns, particularly within the context of the 130-30 trading strategy.

Our paper focuses on providing a comprehensive study comparing different state-of-the-art neural network models, including LSTM, transformer and TCN, for stock prediction using a momentum strategy. We also validate the performance across different markets (S&P 500 for the US market and DAX for the German market). Our model

DAX Return Performance Characteristics												
Equal Weight												
Model	**LSTM**	**TCN**	**TF**	**FNN**	**RF**	**LSTM**	**TCN**	**TF**	**FNN**	**RF**	**RND 1**	**RND 2**
	Before transaction cost					**After transaction cost**					**Market**	
Mean Return	0.011	0.009	0.006	0.017	0.014	0.009	0.003	0.001	0.010	0.009	-0.008	-0.023
Standard Dev.	0.681	0.649	0.678	0.680	0.612	0.681	0.648	0.677	0.680	0.612	1.566	1.593
Minimum	-2.588	-3.159	-3.687	-2.990	-2.468	-2.590	-3.165	-3.693	-2.996	-2.475	-14.53	-29.621
Quartile 1	-0.400	-0.361	-0.357	-0.406	-0.344	-0.401	-0.366	-0.361	-0.413	-0.350	-8.15	-0.825
Median	0.006	0.020	0.022	0.002	0.014	0.004	0.014	0.015	-0.003	0.009	0.000	-0.0217
Quartile 3	0.400	0.381	0.395	0.418	0.377	0.398	0.376	0.390	0.410	0.373	0.805	0.777
Maximum	2.801	4.958	3.562	3.941	3.231	2.800	4.952	3.558	3.934	3.224	16.575	29.621
Return p.a	2.202	1.835	1.337	3.509	2.995	1.829	0.526	0.239	2.122	1.821	-0.024	0.493
Excess Return p.a	2.102	1.735	1.236	3.408	2.895	1.728	0.425	0.139	2.022	1.721	-0.124	0.393
Standard Dev. p.a.	0.653	0.610	0.655	0.679	0.585	0.653	0.610	0.655	0.679	0.585	1.445	1.568
Sharpe Ratio p.a.	4.251	2.372	2.295	4.639	4.644	3.686	0.318	0.631	2.529	2.672	-0.096	0.237
130-30 Trading												
Mean Return	0.057	0.066	0.069	0.065	0.068	0.056	0.061	0.065	0.057	0.064	0.024	0.008
Standard Dev.	1.375	1.284	1.274	1.291	1.160	1.375	1.284	1.274	1.291	1.166	1.492	1.442
Minimum	-5.754	-6.941	-5.481	-6.331	-5.686	-5.755	-6.941	-5.481	-6.336	-5.690	18.892	-32.060
Quartile 1	-0.641	-0.575	-0.619	-0.635	-0.539	-0.641	-0.575	-0.619	-0.640	-0.543	-0.459	-0.427
Median	0.098	0.095	0.048	0.092	0.087	0.097	0.089	0.045	0.087	0.082	0.000	-0.009
Quartile 3	0.845	0.780	0.815	0.787	0.712	0.843	0.777	0.813	0.782	0.707	0.472	0.415
Maximum	6.192	4.909	4.472	6.087	4.960	6.191	4.909	4.468	6.080	4.956	21.547	38.508
Return p.a	11.832	13.708	14.345	13.574	14.222	11.645	12.668	13.482	12.461	13.309	0.493	0.001
Excess Return p.a	11.732	13.608	14.245	13.474	12.791	11.545	12.568	13.381	12.361	13.20	0.016	-0.100
Standard Dev. p.a.	1.236	1.160	1.174	1.194	1.044	1.236	1.160	1.174	1.194	1.044	-0.084	1.328
Sharpe Ratio p.a.	12.309	11.580	12.187	11.106	12.661	12.159	10.708	11.515	10.175	11.807	-0.067	-0.083

Table 9: Performance characteristics of the k=10 equal-weight portfolio (top) and 130-30 portfolio (bottom) for DAX from Jan 2012 to Nov 2017. The first part represents the daily return statistic, while the second part represents the per annual return statistic.

evaluation does not conclude definitive results regarding the performance of any specific models for time series prediction. In general, our models exhibit accuracy ranging from 50% to 52%, aligning with previous studies (Fischer and Krauss, 2018). For the S&P 500 dataset, LSTM demonstrates the highest accuracy across most portfolio sizes (particularly for $k \in \{10, 50, 100, 150, 200\}$), except for k=5, where random forest outperformed with an accuracy of 51.7%. Conversely, for the DAX dataset, the transformer performs well in predicting smaller portfolio sizes (specifically for $k \in \{2, 4, 5\}$), achieving accuracy rates ranging from 51% to 52%. However, for larger portfolio sizes (specifically for $k \in \{8, 10, 15\}$), the Random Forest model emerged as the top performer, maintaining an accuracy level of approximately 51%. We attribute the challenges encountered in achieving conclusive results to the heightened noise present in financial time series data. Considering computational power, time consumed, and model accuracy, we find no reason to prefer neural network models over simple random forests for prediction in our setup.

In our paper, we also validate the applicability of machine learning techniques in stock prediction with a momentum strategy. We observe statistically and economically significant returns by employing an ensembling approach with k=10, where we buy the top 10 winners and short the bottom 10 losers. For the S&P 500 dataset, employing the equal-weighted trading strategy yielded a 2.7% daily return after transaction costs, while the 130-30 trading strategy generated an impressive 9.9% return, both utilizing the TCN model. For the DAX dataset, equal-weighted trading produced a daily return of 0.9% after transaction costs using LSTM and RF predictions. Furthermore, employing the transformer prediction resulted in an average daily return of 6.5%. Evaluating both trading strategies, it becomes apparent that almost all models outperform the random market portfolio over the long term based on average cumulative return per stock. Compared to the random market portfolio, all models exhibit lower standard deviations and less extreme minimum and maximum returns for both equal-weight trading and 130-30 trading strategies, suggesting reduced volatility in their performance. This enhanced stability underscores the effectiveness of the models in managing risk and delivering more consistent performance outcomes compared to choosing stocks randomly. These findings

underscore the efficacy of machine learning methodologies, particularly the momentum strategy, in generating profitable returns in stock prediction tasks.

Lastly, we test out the application of different trading strategies within the momentum strategy framework. Specifically, we compare the performance of equal-weight trading and 130-30 trading strategies in terms of returns and risk. Our analysis reveals that while the 130-30 trading strategy entails increased risk, as indicated by the higher standard deviation resulting from a less diversified portfolio, it also generates significantly higher average returns. This suggests that the 130-30 trading strategy effectively leverages the momentum strategy to capitalize on market opportunities despite the heightened level of risk involved.

Overall, we hope to contribute to bridging the gaps in the existing literature by conducting a comprehensive analysis of state-of-the-art neural network models across different markets using the momentum strategy with different trading strategies.

6 Limitation and Future Work

However, our study has several limitations that merit acknowledgement. Firstly, we rely solely on time-series daily return data to predict whether stocks will outperform the cross-sectional median on the following day. It is important to recognize that time series data alone may be inherently noisy and may not encompass all influential factors affecting our target variables. To address this limitation, we recommend that future research integrates additional features into stock prediction models, such as related news, trading volume, and other relevant factors. By incorporating a more comprehensive set of features, we can potentially enhance the accuracy and robustness of our predictions regarding stock performance, thereby gaining deeper insights into market dynamics and improving the efficacy of our predictive strategies. Therefore, we recommend that investors undertake thorough backtesting and validation processes to assess the robustness and efficacy of predictive models before deploying them in real-world trading scenarios. Moreover, our study focuses on a period without economic recession from 2012 to 2017, in which the stock market is relatively stable. However, it would be valuable to assess the performance of machine learning models during economic downturns, such as the COVID-19 pandemic period in 2019, to evaluate their resilience and effectiveness under different market conditions. Lastly, it's important to note that our study employs a daily trading frequency, which incurs relatively high transaction costs due to frequent adjustments to the portfolio. In future work, exploring different trading frequencies, such as monthly trading, could provide insights into the impact of transaction costs on overall profitability and the effectiveness of predictive models. This could help optimize trading strategies by finding a balance between transaction costs and predictive accuracy.

A Appendix A: List of literature for stock price prediction

Authors	Dataset	Input Variables	Included Models	Evaluation	Best models
Xiang and Wang (2023)	CSI 300 (2005 to 2020)	intraday returns, returns in relation to last closing price, return in relation to opening price	RF, LSTM, TCN	Mean Return and Cumulative Return	TCN
Zhan et al. (2022)	CSI 300 ETF, CSI 300 IF (28 May, 2012 to 30 Dec, 2020)	trading signals from decentralized price spread	logistic regression, XGBoost, CNN, LSTM	Confusion Matrix, ROI and Sharpe Ratio	LSTM
Dai, An & Long (2022) Dai et al. (2022)	SZSE 100	Price Change (classified into 5 category)	LSTM, TCN and TCN (attention)	recall, precision, accuracy	TCN and TCN (attention)
Wang et al. (2022)	CSI 300,S&P 500, N225, HSI (Jan 1, 2010 to Dec 31, 2020)	previous days normalized closing price	Transformer, LSTM, RNN, CNN	MAE, MSE, MAPE	Transformer
Hu (2021)	S&P 500 and Google Data (January 4th, 2010 to May 31st, 2021)	high, low, open, close, volume, adj close	SVM, LSTM, TFT	MAE, SMAPE	TFT
Li et al. (2022)	3 Github dataset that include 145 different stocks and Tweet data (2014 to 2019)	date, movement percent of stock and text on Tweet	Transformer encoder attention (TEA), ARIMA, TSLDA, HAN, CH-RNN, StockNet, Adv-LSTM, CapTE, SMPN, Model1, CNN-BiLSTM-AM, LSTM-RGCN, FOCUS, MAFN	Accuracy, the Matthews correlation coefficient	TEA

Rady et al. (2021)	monthly gold prices (Nov-1989 to Dec-2019)	monthly gold prices	ARIMA, decision tree, random forest, gradient boosted tree	RMSE	random forest
Ma (2020)	Dell (Jan 2010 - Dec 2010)	trading volume, trend, momentum and volatility	ARIMA, ANN, LSTM	accuracy	LSTM
Cao and Wang (2019)	HSI, TSEC, DAX, NASDAQ, S&P 500 (1 March 1990 to 6 January 2014),		CNN, CNN-SVM hybrid model	RMSE, correlation coefficient	CNN-SVM hybrid model.
Krauss et al. (2017)	Companies in S&P 500 (January 1990 to October 2015)	Daily closing price for forecast probability t outperform the market medium	Random Forest (RAF), Gradient Boost, Deep Neural Network (DNN)	Daily Mean Returns	DNN
Khaidem et al. (2016)	AAPL, GE dataset and Samsung Electronics	technical indicators extracted from time series data (Relative Strength Index (momentum indicator), stochastic oscillator, William %R, Moving Average Convergence Divergence, Price Rate of Change)	Random Forest, SVM and Quadratic Discriminant Analysis, logistic regression	out-of-bag error rate, accuracy	Random Forest
Takeuchi and Lee (2013)	individual stocks in NYSE, AMEX or Nasdaq, but excluded stocks with monthly closing prices below $5 per share (Jan 1965 to Dec 2009)	12 past monthly returns and 20 daily return	Deep Learning Model	Confusion matrix for classifying whether stocks outperform the market, average monthly returns	not applicable
Kim and Kang (2019)	KOSPI 200 with currency, global index, and commodities index parameters (2017 to 2018)	change ratio of KOSPI	multilayer perceptron (MLP), CNN), stacked LSTM, attention LSTM, and weighted attention LSTM	hit ratio	attention LSTM

Zhou et al. (2019)	Nasdaq (3 Jan 2012 to 23 Dec 2016), S&P 500 (3 Jan 2012 to 23 Dec 2016), SSEC (4 Jan 2010 to 31 Dec 2014)	daily open price, high price, low price, close price and trading volume	logistic regression, gradient boosted tree, LR2GBDT	Average annual return, maximum drawdown, Sharpe Ratio, Average Annualised Return, Number of Trades	LR2GBDT
Karmiani et al. (2019)	9 companies (Apple, Acer, Amazon, Google, HP, IBM, Intel, Microsoft, Sony) (Jan 2009 to Oct 2018)	Open, high, low, close, adj close, volume,	Backpropagation, SVM & LSTM	accuracy, SD, variance, time, t-test	LSTM
Kim and Kim (2019)	S&P 500 ETF data (14 Oct 2016 to 16 Oct 2017)	high, low, open, close price and volume data	stock time series LSTM (ST-LSTM), stock chart CNN (SC-CNN), LSTM-CNN (combine both LSTM and CNN)	RMSE, MAE, MAPE	LSTM-CNN
Sethia and Raut (2019)	S&P500 index price (March 2000 to Oct 2017)	50 technical indicators, such as Open, high, low, close, volume, Daily returns, 14-day Bollinger bands, etc.	LSTM, GRU, ANN and SVM	RMSE, R2 score, Return ratio	LSTM
Fischer and Krauss (2018)	all companies listed on S&P 500 (Dec 1989 to Sept 2015)	1000 days daily return sequence (750 days for training, 250 days for testing)	LSTM, RAF, DNN, Logistic regression	Daily Return, Sharpe ratio Sortino ratio	LSTM

Siami-Namini et al. (2018)	Index data: N225, IXIC, HIS, GSPC, DJ (Jan 1985 to Aug 2018); Monthly economics Data: Medical care commodities(Jan 1967 to July 2017), Housing for all urban consumers (Jan 1967 to July 2017), the trade-weighted U.S. dollar (Aug 1967 to July 2017), Food and Beverages for all urban consumers (Jan 1967 to July 2017), M1 Money Stock (Jan 1959 to July 2017), Transportation for all urban consumers (Jan 1947 to July 2017)	adjusted closed price	ARIMA, LSTM	RMSE	LSTM
Althelaya et al. (2018)	S&P 500 index (01/01/2010 to 30/11/2017)	high, low, open, close and volume data	stacked LSTM (SLSTM), stacked GRU (SGRU), bidirectional LSTM (BLSTM), and bidirectional GRU (BGRU)	RMSE, MAE, MAPE	stacked LSTM (SLSTM)

Zhong and Enke (2017b)	S&P 500 ETF (1 Jun 2003 to 31 May 2013)	SPY return for the current day and three previous days, the relative difference in the percentage of the SPY return, exponential moving averages of the SPY return, Treasury bill (T-bill) rates, certificate of deposit rates, financial and economic indicators, the term and default spreads, exchange rates between the USD and four other currencies, the return of seven world major indices (other than the S&P 500), SPY trading volume, and the return of eight large capitalization companies within the S&P 500 (which is a market cap-weighted index and driven by larger capitalization companies)	ANN using dimensionality reduction including PCA, FRPCA and KPCA	MSE and confusion matrix, daily return, Sharpe ratio	ANN-PCA
Dey et al. (2016)	Apple Inc.	Technical indicators: Relative strength index, Stochastic Oscillators, William % R, Moving Average Convergence Divergence (MACD), Price rate of Change, Volume	logistic regression, SVM, RF, ANN, XGBoost	direction of stocks on the following day	XGBoost
Shen et al. (2012)	NASDAQ, S¶500, DJIA, Nikkei 225 (4 Jan 2000 to 25 Oct 2012)	world major stock indices, including stocks, currency and commodity	SVM, multiple additive regression trees (MART) with different window size	confusion matrix, RMSE	Weature selection: SVM. Without feature selection: MART

Kara et al. (2011)	ISE National 100 Index (2 Jan 1997 - 31 Dec 2007)	10 technical indicators (Simple 10-day moving average, Weighted 10-day moving average, Momentum Stochastic K%, Stochastic D%, RSI (Relative Strength Index), MACD (moving average convergence divergence), Larry William's R%, A/D (Accumulation/Distribution) Oscillator, CCI (Commodity Channel Index))	SVM & ANN	accuracy	ANN
Bao et al. (2004)	Haier (15 Apr 2003 to 25 Nov 2003)	Relative difference in percentage change of daily closing price (RDP-5, RDP-10, RDP-15 and RDP-20)	SVM	RDP+5	SVM
Kim (2003)	KOSPI	12 technical indicators (&K, %D, Slow %D, Momemtum, ROC, Williams' %R, A/D Oscillator, Disparity5, Disparity10, CCI, RSI)	SVM, ANN and case-based reasoning	Hit Ratio	SVM
Dai, An and Long (2002) Dai et al. (2022)	constituent stocks of SZSE 100 (Dec 2016)	Ultra High Frequency (UHF) Price Changes	SGRACH, LSTM, TCN and TCN (attention)	Confusion Matrix	TCN
Tino et al. (2001)	DAX (22 August 1991 to 8 June 1998), FTSE 100 (29 May 1991 to 29 December 1995)	Daily closing price, Daily closing prices of call and put options	Markov Model, GARCH, Markov Model, Elman RNN	Daily Mean Returns	Elman RNN and Markov

Leung et al. (2000)	S&P 500, FTSE 100 and Nikkei 225	classification models (Linear discriminant analysis, logit, probit, probabilistic neural networks), level estimation models (adaptive exponential smoothing, vector autoregression with Klaman filter, multivariate transfer function, multivariate transfer function, multilayered feed-forward neural network)	Short-term interest rate, long-term interest rate, lagged index return, consumer price level and industrial production level	Return on index, probability given the direction of return	Classification models

Table 10: List of literature for stock price prediction

B Appendix B: Cumulative Average Return

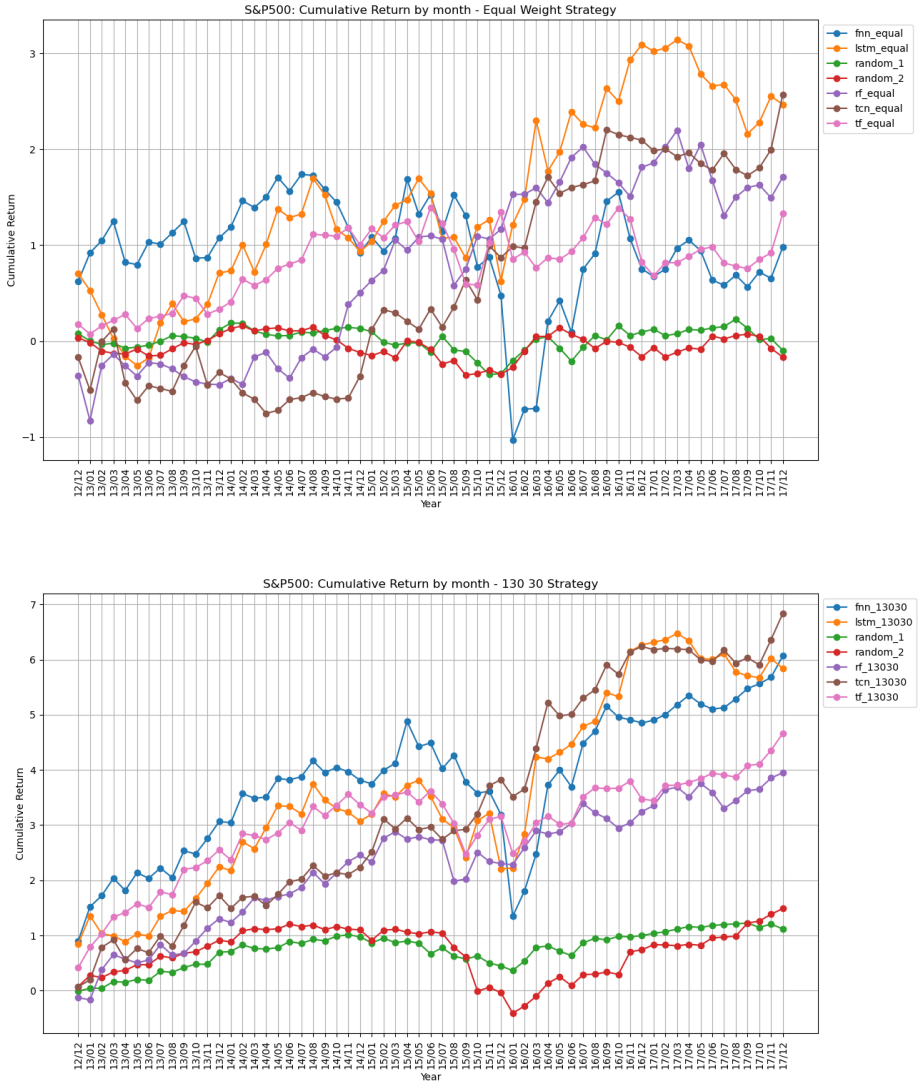

Figure 6: Cumulative Return for S&P500 with equal weight strategy (top) and 130-30 weight strategy (bottom). It represents, in average, how much return a stock is going to generate on 1 USD investment per day during the whole prediction period 2012-2017

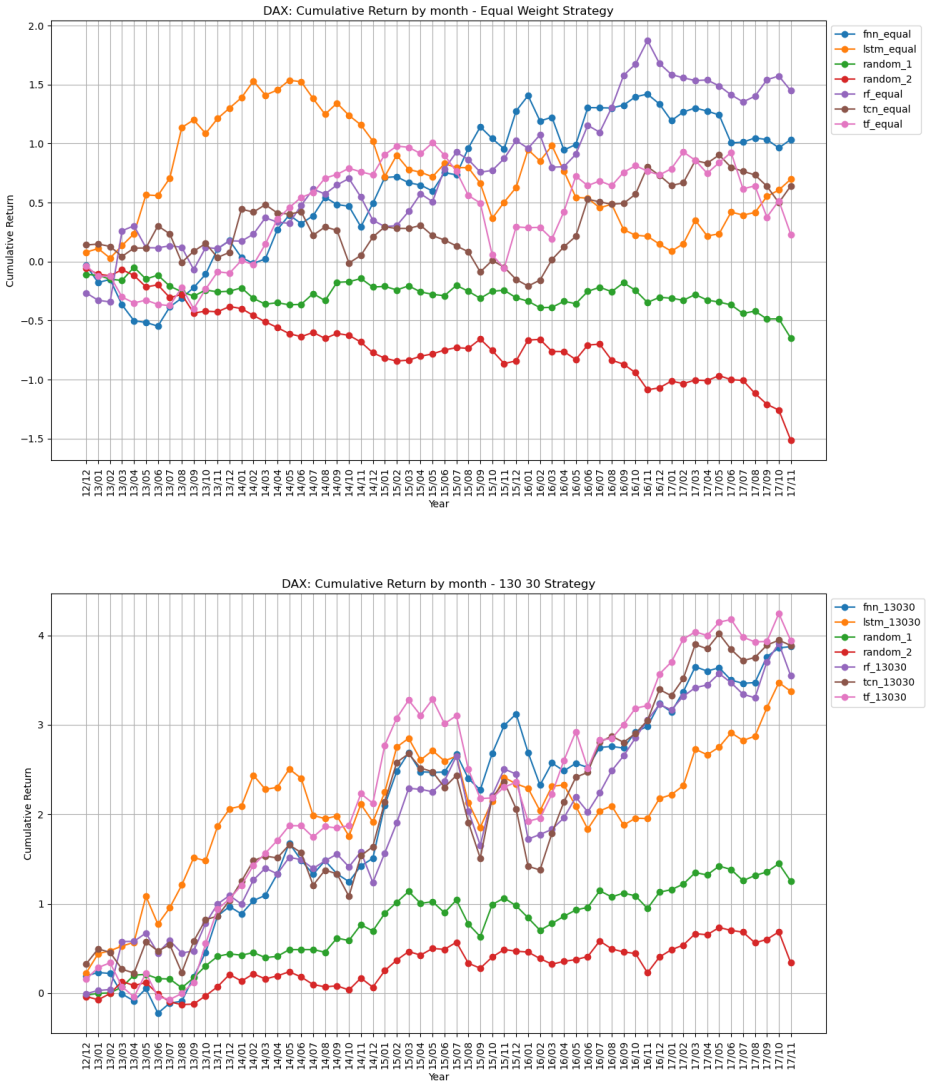

Figure 7: Cumulative Return for DAX with equal weight strategy (top) and 130-30 weight strategy (bottom). It represents,in average,how much return a stock is going to generate on 1 USD investment per day during the whole prediction period 2012-2017

References

Abate, G., Bonafini, T., and Ferrari, P. (2022). Portfolio constraints: An empirical analysis. *International Journal of Financial Studies*, 10(1):9.

Adebayo, F. A., Sivasamy, R., and Shangodoyin, D. K. (2014). Forecasting stock market series with arima model. *Journal of Statistical and Econometric Methods*, 3(3):65–77.

Althelaya, K. A., El-Alfy, E.-S. M., and Mohammed, S. (2018). Stock market forecast using multivariate analysis with bidirectional and stacked (lstm, gru). In *2018 21st Saudi Computer Society National Computer Conference (NCC)*, pages 1–7. IEEE.

Ariyo, A. A., Adewumi, A. O., and Ayo, C. K. (2014). Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th international conference on computer modelling and simulation*, pages 106–112. IEEE.

Attigeri, G. V., MM, M. P., Pai, R. M., and Nayak, A. (2015). Stock market prediction: A big data approach. In *TENCON 2015-2015 IEEE Region 10 Conference*, pages 1–5. IEEE.

Avellaneda, M. and Lee, J.-H. (2010). Statistical arbitrage in the us equities market. *Quantitative Finance*, 10(7):761–782.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Bao, Y., Lu, Y., and Zhang, J. (2004). Forecasting stock price by svms regression. In *Artificial Intelligence: Methodology, Systems, and Applications: 11th International Conference, AIMSA 2004, Varna, Bulgaria, September 2-4, 2004. Proceedings 11*, pages 295–303. Springer.

Bebis, G. and Georgiopoulos, M. (1994). Feed-forward neural networks. *Ieee Potentials*, 13(4):27–31.

Biau, G. and Scornet, E. (2016). A random forest guided tour. *Test*, 25:197–227.

Cao, J. and Wang, J. (2019). Stock price forecasting model based on modified convolution neural network and financial time series analysis. *International Journal of Communication Systems*, 32(12):e3987.

Chan, L. K., Jegadeesh, N., and Lakonishok, J. (1999). The profitability of momentum strategies. *Financial Analysts Journal*, 55(6):80–90.

Chavas, J.-P. and Holt, M. T. (1993). Market instability and nonlinear dynamics. *American Journal of Agricultural Economics*, 75(1):113–120.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Dai, W., An, Y., and Long, W. (2022). Price change prediction of ultra high frequency financial data based on temporal convolutional network. *Procedia Computer Science*, 199:1177–1183.

Deboeck, G. J. (1994). *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets*, volume 39. John Wiley & Sons.

Dey, S., Kumar, Y., Saha, S., and Basak, S. (2016). Forecasting to classification: Predicting the direction of stock market price using xtreme gradient boosting. *PESIT South Campus*, pages 1–10.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Enke, D. and Thawornwong, S. (2005). The use of data mining and neural networks for forecasting stock market returns. *Expert Systems with applications*, 29(4):927–940.

Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European journal of operational research*, 270(2):654–669.

Fjellström, C. (2022). Long short-term memory neural network for financial time series. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 3496–3504. IEEE.

Fletcher, T. (2009). Support vector machines explained. *Tutorial paper*, 1118:1–19.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378.

Frino, A., Jarnecic, E., and Das, A. (2008). The performance of the 130/30 strategy in the australian equities market. In *21st Australasian Finance and Banking Conference*.

Gandhmal, D. P. and Kumar, K. (2019). Systematic analysis and review of stock market prediction techniques. *Computer Science Review*, 34:100190.

Graves, A. and Graves, A. (2012). Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45.

Grinold, R. C. and Kahn, R. N. (2000). The efficiency gains of long–short investing. *Financial Analysts Journal*, 56(6):40–53.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hsu, M.-W., Lessmann, S., Sung, M.-C., Ma, T., and Johnson, J. E. (2016). Bridging the divide in financial market forecasting: machine learners vs. financial economists. *Expert systems with Applications*, 61:215–234.

Hu, X. (2021). Stock price prediction based on temporal fusion transformer. In *2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, pages 60–66. IEEE.

Huck, N. (2009). Pairs selection and outranking: An application to the s&p 100 index. *European Journal of Operational Research*, 196(2):819–825.

Ican, O., Celik, T. B., et al. (2017). Stock market prediction performance of neural networks: A literature review. *International Journal of Economics and Finance*, 9(11):100–108.

Idrees, S. M., Alam, M. A., and Agarwal, P. (2019). A prediction approach for stock market volatility based on time series data. *IEEE Access*, 7:17287–17298.

Investopedia (n.d.). What is a good sharpe ratio? `https://www.investopedia.com/ask/answers/010815/what-good-sharpe-ratio.asp`.

Jegadeesh, N. and Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of finance*, 48(1):65–91.

Jegadeesh, N. and Titman, S. (2001). Profitability of momentum strategies: An evaluation of alternative explanations. *The Journal of finance*, 56(2):699–720.

Jegadeesh, N. and Titman, S. (2023). Momentum: Evidence and insights 30 years later. *Pacific-Basin Finance Journal*, page 102202.

Johnson, G., Ericson, S., and Srimurthy, V. (2007). An empirical analysis of 130/30 strategies. *Journal of Alternative Investments*, 10(2):31.

Kara, Y., Boyacioglu, M. A., and Baykan, Ö. K. (2011). Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. *Expert systems with Applications*, 38(5):5311–5319.

Karmiani, D., Kazi, R., Nambisan, A., Shah, A., and Kamble, V. (2019). Comparison of predictive algorithms: backpropagation, svm, lstm and kalman filter for stock market. In *2019 amity international conference on artificial intelligence (AICAI)*, pages 228–234. IEEE.

Khaidem, L., Saha, S., and Dey, S. R. (2016). Predicting the direction of stock market prices using random forest. *arXiv preprint arXiv:1605.00003*.

Kim, K.-j. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319.

Kim, S. and Kang, M. (2019). Financial series prediction using attention lstm. *arXiv preprint arXiv:1902.10877*.

Kim, T. and Kim, H. Y. (2019). Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. *PloS one*, 14(2):e0212320.

Kirisci, M. and Cagcag Yolcu, O. (2022). A new cnn-based model for financial time series: Taiex and ftse stocks forecasting. *Neural Processing Letters*, 54(4):3357–3374.

Klioutchnikov, I., Sigova, M., and Beizerov, N. (2017). Chaos theory in finance. *Procedia computer science*, 119:368–375.

Krauss, C., Do, X. A., and Huck, N. (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500. *European Journal of Operational Research*, 259(2):689–702.

Kumbure, M. M., Lohrmann, C., Luukka, P., and Porras, J. (2022). Machine learning techniques and data for stock market forecasting: A literature review. *Expert Systems with Applications*, 197:116659.

Kurani, A., Doshi, P., Vakharia, A., and Shah, M. (2023). A comprehensive comparative study of artificial neural network (ann) and support vector machines (svm) on stock forecasting. *Annals of Data Science*, 10(1):183–208.

Leung, M. T., Daouk, H., and Chen, A.-S. (2000). Forecasting stock indices: a comparison of classification and level estimation models. *International Journal of forecasting*, 16(2):173–190.

Lewellen, J. (2002). Momentum and autocorrelation in stock returns. *The Review of Financial Studies*, 15(2):533–564.

Li, A. W. and Bastos, G. S. (2020). Stock market forecasting using deep learning and technical analysis: a systematic review. *IEEE access*, 8:185232–185242.

Li, Q., Jiang, L., Li, P., and Chen, H. (2015). Tensor-based learning for predicting stock movements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Li, Y., Lv, S., Liu, X., and Zhang, Q. (2022). Incorporating transformers and attention networks for stock movement prediction. *Complexity*, 2022:1–10.

Lim, B., Arık, S. Ö., Loeff, N., and Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764.

Lo, A. W. (2010). *Hedge funds: An analytic perspective*. Princeton University Press.

Lo, A. W., Mamaysky, H., and Wang, J. (2000). Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The journal of finance*, 55(4):1705–1765.

Lo, A. W. and Patel, P. N. (2008). 130/30: The new long-only. *INSTITUTIONAL INVESTOR-NEW YORK-*, 42(5):186.

Ma, Q. (2020). Comparison of arima, ann and lstm for stock price prediction. In *E3S Web of Conferences*, volume 218, page 01026. EDP Sciences.

Maillard, S., Roncalli, T., and Teïletche, J. (2010). The properties of equally weighted risk contribution portfolios. *Journal of Portfolio Management*, 36(4):60.

Makridakis, S. and Hibon, M. (1997). Arma models and the box–jenkins methodology. *Journal of forecasting*, 16(3):147–163.

Moritz, B. and Zimmermann, T. (2016). Tree-based conditional portfolio sorts: The relation between past and future stock returns. *Available at SSRN 2740751*.

Nti, I. K., Adekoya, A. F., and Weyori, B. A. (2020). A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review*, 53(4):3007–3057.

O'shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Parmar, A., Katariya, R., and Patel, V. (2019). A review on random forest: An ensemble classifier. In *International conference on intelligent data communication technologies and internet of things (ICICI) 2018*, pages 758–763. Springer.

Rady, E., Fawzy, H., and Fattah, A. M. A. (2021). Time series forecasting using tree based methods. *J. Stat. Appl. Probab*, 10(1):229–244.

Reddy, C. V. (2019). Predicting the stock market index using stochastic time series arima modelling: The sample of bse and nse. *Indian Journal of Finance*, 13(8):7–25.

Rigatti, S. J. (2017). Random forest. *Journal of Insurance Medicine*, 47(1):31–39.

Rouwenhorst, K. G. (1998). International momentum strategies. *The journal of finance*, 53(1):267–284.

Schwartz, R. A. and Whitcomb, D. K. (1977). The time-variance relationship: Evidence on autocorrelation in common stock returns. *The Journal of Finance*, 32(1):41–55.

Selvin, S., Vinayakumar, R., Gopalakrishnan, E., Menon, V. K., and Soman, K. (2017). Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE.

Sethia, A. and Raut, P. (2019). Application of lstm, gru and ica for stock price prediction. In *Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2018, Volume 2*, pages 479–487. Springer.

Sharpe, W. F. (1998). The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 3:169–185.

Shen, S., Jiang, H., and Zhang, T. (2012). Stock market forecasting using machine learning algorithms. *Department of Electrical Engineering, Stanford University, Stanford, CA*, pages 1–5.

Siami-Namini, S., Tavakoli, N., and Namin, A. S. (2018). A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1394–1401. IEEE.

Song, Y.-Y. and Ying, L. (2015). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130.

Sorensen, E. H., Hua, R., and Qian, E. (2006). Aspects of constrained long-short equity portfolios. *Journal of Portfolio Management*, 33(2):12.

Takeuchi, L. and Lee, Y.-Y. A. (2013). Applying deep learning to enhance momentum trading strategies in stocks. In *Technical Report*. Stanford University Stanford, CA, USA.

Tino, P., Schittenkopf, C., and Dorffner, G. (2001). Financial volatility trading using recurrent neural networks. *IEEE transactions on neural networks*, 12(4):865–874.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Vilela, L. F., Leme, R. C., Pinheiro, C. A., and Carpinteiro, O. A. (2019). Forecasting financial series using clustering methods and support vector regression. *Artificial Intelligence Review*, 52:743–773.

Wang, C., Chen, Y., Zhang, S., and Zhang, Q. (2022). Stock market index prediction using deep transformer model. *Expert Systems with Applications*, 208:118128.

Xiang, X. and Wang, W. (2023). Predicting intraday trading direction of csi 300 based on tcn model. In *2023 2nd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*, pages 293–299. IEEE.

Yamada, Y., Suzuki, E., Yokoi, H., and Takabayashi, K. (2003). Decision-tree induction from time-series data based on a standard-example split test. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 840–847.

Yen, G. and Lee, C.-f. (2008). Efficient market hypothesis (emh): past, present and future. *Review of Pacific Basin Financial Markets and Policies*, 11(02):305–329.

Zhan, B., Zhang, S., Du, H. S., and Yang, X. (2022). Exploring statistical arbitrage opportunities using machine learning strategy. *Computational Economics*, 60(3):861–882.

Zhang, M., Tang, X., Zhao, S., Wang, W., and Zhao, Y. (2022). Statistical arbitrage with momentum using machine learning. *Procedia Computer Science*, 202:194–202.

Zhong, X. and Enke, D. (2017a). A comprehensive cluster and classification mining procedure for daily stock market return forecasting. *Neurocomputing*, 267:152–168.

Zhong, X. and Enke, D. (2017b). Forecasting daily stock market return using dimensionality reduction. *Expert systems with applications*, 67:126–139.

Zhou, F., Zhang, Q., Sornette, D., and Jiang, L. (2019). Cascading logistic regression onto gradient boosted decision trees for forecasting and trading stock indices. *Applied Soft Computing*, 84:105747.