Lasse Lilius

Forecasting Finnish stock market using machine learning methods

School of Accounting and Finance
Master's thesis in Economics
Master Programme of Economics

Vaasa 2024

UNIVERSITY OF VAASA
School of Accounting and Finance
Author: Lasse Lilius
Title of the Thesis: Forecasting Finnish stock market using machine learning methods
Degree: Kauppatieteiden maisteri
Programme: Taloustiede
Supervisor: Lasse Lilius
Year: 2024 **Sivumäärä:** 54

ABSTRACT:
Tutkielmassa tarkastellaan Helsingin pörssin osakeindeksin OMXH25 ennustettavuutta koneoppimismalleilla. Useimmat koneoppimismallit on kehitetty vuosikymmeniä sitten, mutta viime vuosikymmenten jatkuva kehitys tietokoneiden laskentatehossa on mahdollistanut näiden mallien laajemman hyödynnettävyyden. Tutkielmassa käytetyt koneoppimismallit ovat k-nearest neighbors, random forest classifier, gradient boosting classifier, support vector machine ja artificial neural network.

Tutkielmassa ennustetaan OMXH25 osakeindeksin seuraavan päivän hinnan kehitystä. Päivittäiset hinnan kehitykset on jaettu kolmeen luokkaan, jossa alimmassa luokassa ovat suurimmat negatiiviset muutokset, ylimmässä luokassa ovat suurimmat positiiviset muutokset ja keskimmäisessä luokassa ovat mediaanin ympärillä olevat hinnan kehitykset. Luokkajaottelun myötä ennustamme seuraavan päivän hinnan kehityksen luokkaa, hinnan tarkan arvon sijasta.

Tutkielmassa kerättiin 8014 päivittäistä havaintoa vuodesta 1991 vuoteen 2022. Muuttujia ovat eri maiden pörssien hinnat, raaka-aine markkinoiden hinnat, korkotaso sekä Helsingin pörssin hinnasta laskettuja teknisiä indikaattoreita. Havainnot jaettiin kahteen joukkoon, joista toista käytettiin koneoppimismallien harjoittamiseen ja toista koneoppimismallien testaukseen. Lisäksi mallien ennusteiden avulla luodaan kaupankäyntistrategia ja strategiaan perustuva simulaatio, jonka avulla voidaan mitata mallien kykyä ennustaa osakemarkkinoiden kehitystä.

Kaikki tutkielmassa käytetyt koneoppimismallit pystyivät ennustamaan Helsingin pörssin OMXH25 osakeindeksin seuraavan päivän tuottoja. Koneoppimismallien ennusteiden perusteella luotu kaupankäyntistrategia tuotti paremmin kuin jatkuvaan markkinoilla oloon perustuva vertailuindeksi. Parhaiten tuottava koneoppimismalli oli gradient boosting classifier.

KEYWORDS: machine learning, stock market, predictability, forecasting, finance

Contents

Figures

Tables

Algorithms

1 Introduction

Stock market forecasting is of interest in the financial economics as profits can be made from correct forecasts. The predictability of the stock market has been debated for a long time. Fama (1965) presented evidence supporting random walk hypothesis which states that price level of stocks is random and cannot be predicted. Furthermore, Fama (1970) introduced the efficient market hypothesis and evidence supporting it. According to the weak form of efficient market hypothesis the price of a stock cannot be predicted from previous price movements. In addition, in the semi-strong form of efficient market hypothesis the effect of new public information, quarterly earnings of a company for example, translates efficiently to the price of the stock. In the strong-form of the hypothesis also private information is taken into account. According to Malkiel (2003) there might be short departures where the stock market strays away from efficiency, but these occurences are the exception rather than the rule.

On the contrary, Lee et al. (2010) studied the efficient market hypothesis in 32 developed and 26 developing countries between years 1999 and 2007 and concluded that stock markets are inefficient. Furthermore, Hamid et al. (2017) found that the Asia-Pacific stock markets are not efficient and Dias et al. (2020) concluded that during the covid-19 pandemic the US, Chinese and Portugese stock markets were inefficient. In addition, Neely et al. (2014) found combining technical indicators and macroeconomic variables produces superior forecasting power of monthly equity risk premium.

The analysis of stock market can be divided into two categories, technical and fundamental analysis. Technical analysis focuses on stocks price history and uses variety of indicators calculated from the price history. The aim is to find patterns which can be used to predict stock prices. Examples of indicators in technical analysis are moving averages and momentum indicators. Fundamental analysis on the other hand focuses on macroeconomic environment, industry specific outlook and companys financial performance and variables derived from the financial performance.

Fundamental analysis indicators include for example price-to-earnings ratio and liquidity ratios like current ratio.

Algorithms used in machine learning have been invented decades ago, but the surge in computing power has enabled the use of these methods in broader scale. Moore's Law, which states that the performance of digital electronics will double roughly every two years, has underpinned the IT industry for the last 50 years (Shalf, 2020). As a result, the computing power has risen multifold during this period. Early examples of machine learning and artificial intelligence were in board games, where in 1997 chess computer Deep Blue beat chess world champion Gary Kasparov in a chess match. Later in 2015 and 2016 computer program AlphaGo was able to beat professional players in Go, which is a more complicated board game compared to Chess.

In the early years of 2020s generative artificial intelligence has gained ground. Generative artificial intelligence refers to large language models which are able to generate for example text and video from prompts given by users. Perhaps the best-known generative artificial intelligence being CHATGPT a chat bot created by OpenAI. OpenAI has also created text-to-picture generator DALL-E as well as text-to-video model SORA. In february 2024 OpenAI received an 80 billion US dollar valuation, depicting the surge in demand for generative artificial intelligence (Metz & Mickle, 2024).

The aim of this study is to predict the daily returns of the Helsinki stock market using machine learning methods and to measure the accuracy of these predictions. Weighted stock market index OMXH25 will represent the Helsinki stock market, consisting of the 25 most traded stocks. Dataset of 8014 daily observations between 1991 and 2022 are gathered and machine learning methods are trained using this dataset.

This study closely follows and builds on Nevasalmi (2020) where the author forecasted the daily returns of S&P500 using machine learning. In this study as well as in the Nevasalmi (2020), the predicion setting is transformed into classification setting where the predicted value is the direction of stock market, rather than the actual continuous value of a stock price. Furthermore the binary classification setting is transformed into multinomial setting of three classes where the first class implies low returns, second class implies close to median returns and third class implies large returns. Multinomial setting allows to predict the large negative and positive returns which affect trading profitability more than the close to median daily returns.

This study is structured as follows. In section 2 the literature regarding stock market prediction usign machine learning methods is reviewed. In section 3 the used machine learning methods are presented and in section 4 the used dataset is presented. Optimization of hyperparameters is performed in section 5 while the statistic and economic predictive performance are presented in section 6. Lastly, the conclusions of the study are presented in section 7.

2 Literature review

In this section I will review the previous literature on forecasting with machine learning methods in the financial markets.

Sudha and Nambi (2012) studied predictability of Indian stock market indices using *k*-nearest neighbors classifier. Data included daily price data of two indexes BSE-SENSEX and NSE-NIFTY between 2006 and 2010 with 1110 trading days. Data was split to training and test data with 80:20 split ratio and target was to predict if next day had positive or negative returns. *k*-nearest neighbors classifier was trained with the training dataset and the predictive performance of the model was measured with the test data. As a result the model reached 88,74 percent and 79,65 percent accuracies for the indices, with similar accuracy in positive and negative return predictions.

Teixeira and De Oliveira (2010) studied automatic stock trading based on *k*-nearest-neighbor classifier. Daily price observations for fifteen stocks in Sao Paulo stock exchange were gathered between the years 1998 – 2009. The dataset consisted of technical indicators calculated from daily stock prices and the dataset was divided into 10 subsets chronologically. Rolling window was used for training where the previous three subsets were used for training and the following single subset for testing. A trading model is constructed based on classifier to either buy or sell a stock. Model outperformed buy and hold benchmark in 12 of the 15 studied stocks.

Alkhatib et al. (2013) used *k*-nearest neighbors classifier to predict the Jordanian stock market. Five companies from the stock exchange were selected randomly and daily data was collected from June 2009 to December 2009, amounting to approximately 200 daily records. For each day the closing price, highest price and lowest price were gathered and the study used *k* -value of 5. Authors predicted the stock price of next day and concluded that the predicted stock prices were close to the actual prices.

Abdelmoula (2015) found that *k*-nearest neighbor classifier is able to predict commercial loan quality between healthy and risky loans using financial ratios of lender. Gaganis et al. (2007) studied *k*-nearest neighbors classifier to estimate auditors' opinions of companies in UK. Authors found that *k*-nearest neighbors classifier was more efficient than discriminant and logit models in average classification accuracy. Mukid et al. (2018) studied credit scoring using weighted *k*-nearest neighbor classifier, which gives more weight to neighbors close to observation, and found that the classifier would be usable in credit scoring. On the other hand, Soepriyanto (2021) concluded that *k*-nearest neighbors prediction of stock values was poor.

Khan et al. (2016) predicted Karachi Stock Exchange, London Stock Exchange and New York Stock Exchange using machine learning methods. Daily data was gathered for period 2009 to 2014 and the considered machine learning methods were k-nearest neighbors, support vector machine and naïve bayes classifier. Authors conclude that k-nearest neighbors was the best classifier followed by support vector machine.

Khaidem et al. (2016) forecasted stock market using random forest classifier where the target was to classify positive or negative returns during time period. Apple, Microsoft and Samsung stocks were studied and 85-95% accuracy was achieved for long-term prediction which was 30 to 90 trading days.

Basak et al. (2019) studied predicting stock market with tree-based classifiers. Considered classifiers were random forest and gradient boosting implemented with XGBoost library. Time-series data was exponentially smoothed to give more value to recent observations. Data consisted of 10 stocks in the US stock market and each days closing price and trading volume from listing day to 3.2.2017. Technical indicators were calculated based on closing price and included Relative Strength Index (RSI), Stochastic Oscillator, Williams %R and Moving Average Convergence Divergence (MACD). With random forest the accuracy for predictions 3 days into the future ranged from 63,53 to 76,04 and for long-term predictions of 90 days the accuracies were impressive and ranged

from 86,10 to 95,44. For Gradient boosting the accuracies ranged between 55,18 and 71,07 for 3-day period and between 70,5 and 94,44 for 90-day period.

Campisi et al. (2023) compared machine learning methods in predicting the US stock market. The study included both regression and classification models where classification targets were positive or negative returns in the following 30 days. Data centered around market volatility indices like VIX index and included 3040 daily observations from 2011 to 2022. Random forest achieved accuracy of 0,8239 and gradient boosting classifier achieved accuracy of 0,7113.

Ballings et al. (2015) studied the predicative ability of multiple machine learning methods. Methods in the study included k-nearest neighbors, random forest, support vector machine and feed-forward neural network classifiers. Data for 5767 European companies was collected and it included company specific indicators for example price-to-earnings ratio, liquidity indicators like current ratio and profitability indicators like return on equity. Data was collected for year 2009 and the target was to predict one year ahead to year 2010 where if the market price of a stock had risen 25% or more it was classified as positive return and otherwise as negative return. Authors found that random forest outperformed the other classifiers by a wide margin with an area under curve score of 0,9037, while support vector machine achieved score of 0,8395. Feed-forward neural network and k-nearest neighbors achieved scores of 0,7279 and 0,7264 respectively. Scores represent the median of 2-fold cross-validation performed 5 times.

Toochaei and Moeini (2023) studied ensemble classifiers and their performance in stock market prediction. Data consisted of 15 yearly observations for 187 companies listed in Iranian stock market TSE from year 2005 to 2020. The study used company specific financial ratios as well as macroeconomic indicators, while technical indicators were not used. Authors divided stocks to 4 groups, very low, low, high and very high based on their yearly returns and this class was used for prediction. A total of 14 ensemble methods were studied where accuracies ranged from 59,5 to 83,5. Gradient

boosting classifier ranked fourth with accuracy of 80,4 and random forest classifier ranked fifth with accuracy of 79,6. The study was conducted using 10-fold cross-validation.

Roy et al. (2020) compared the forecasting performance of random forest, gradient boosting, and deep neural network in South-Korean stock market. Quarterly time-series data was gathered from 2007 to 2015 for 37 companies including company specific financial report data as well as macroeconomic indicators. The study was a binary classification of whether a stock rose more than 25% in a year. 10-fold cross-validation was performed with accuracy results of 0,82 for random forest, 0,81 for gradient boosting and 0,78 for deep neural network. The studied neural network was a feed-forward neural network and it consisted of three hidden layers.

Lohrman and Luukka (2019) studied forecasting of intraday returns of S&P500 index with random forest classifier. Training dataset consisted of time series from 2010 to 2016 with 1373 daily observations and test dataset from 2016 to 2018 with 481 daily observations. Predictor set included other market indices, currency and interest rate data as well as technical and volatility indicators and authors used filtering methods to extract optimal predictor subset. Target to be forecasted was the daily returns from opening price to close price of S&P500 which were divided to four groups, larger than 0,5%, between 0% and 0,5%, between -0,5% and 0 as well as smaller than -0,5% returns. Random forest classifier was compared with k-nearest neighbors, naïve bayes and decision tree classifiers. As a result, random forest classifier was the best performer, while k-nearest neighbor classifier was the worst performer. In addition, the model performed better in predicting positive returns than negative returns. Four different trading strategies were constructed based on the model and all of the strategies beat the buy and hold benchmark while best strategy returned 21,09 percent annually compared to 12,93 of the benchmark.

Khan et al. (2023) studied stock market prediction with nine different machine learning methods. Machine learning methods included support vector machine, k-nearest neighbors, random forest, gradient boost variant XGBoost and single-layer feed-forward neural network among others. Daily closing prices and price 15 minutes after opening of TSLA stock was gathered for time period 2016 – 2021 and dataset was divided to training and test datasets. A binary classifier was constructed on whether the stock price rises or falls in the next day. Separate models were built for end-of-day price and price 15 minutes after market open. In end of day classification accuracy logistic regression was the best performer with 85,51 followed by XGBoost with 84,8, feed-forward neural network and random forest with 84,45 and k-nearest neighbors ranked eighth with 79,15. Predictions on the 15 minutes after opening price were more accurate as random forest was the best with accuracy of 91,27, XGBoost was second with accuracy of 90.93, feed-forward neural network was fifth with accuracy of 89,93, support vector machine was sixth with 88,59 accuracy and k-nearest neighbors was last with accuracy of 80,53. Financial simulation was carried out based on predictions and random forest performed the best by a wide margin in both end of day and 15 minutes after open scenarios.

Ampomah et al. (2020) examined stock price prediction using decision tree-based ensemble machine learning models including random forest, XGBoost and AdaBoost among others. Eight stocks or stock indices were randomly chosen from US and Indian stock markets and 3774 daily observations were gathered from 2005 to 2019. For the predictor set 40 different technical indicators were calculated including RSI, MACD and moving averages. Random forest achieved mean accuracy of 0,819 which ranked fifth of the six classifiers.

Patel et al. (2015) studied stock price prediction with machine learning methods. Considered classifiers were single-layer feed-forward neural network, support vector machine, random forest as well as naïve bayes classifier. The dataset consisted of 2474 daily observations between 2003 and 2012 for stock indices CNX Nifty and S&P BSE

Sensex and for two stocks Reliance Industries and Infosys Ltd. Gathered technical indicators included moving averages, RSI and MACD. Random forest performed the best of the classifiers when using continuous data with accuracy of 83,56.

Sadorsky (2022) found that random forest and support vector machine achieved prediction accuracy of over 85 when prediction horizon was longer than 10 days when predicting solar stock ETF prices. Dataset included 2717 daily observations from 2011 to 2021 and technical indicators were calculated from daily prices. Additionally, Sadorsky (2021) predicted stock prices of five widely traded clean energy ETFs using machine learning methods in a binary classification setting. The gathered dataset consisted of daily prices between 2009 and 2020 and technical indicators like RSI and MACD. Random forest classifier achieved over 85 accuracy when prediction horizon was 10 days. In addition, Sadorsky (2021) studied the predictability of gold and silver ETFs with 3714 daily observations from 2006 to 2021. Using technical indicators, the prediction accuracy binary classification with random forest and gradient boosting was over 85% on a 10 day horizon.

Liu et al. (2015) studied financial fraud detection using random forest and other classifiers and they observed that random forest had the best classification performance in detecting fraudulent companies.

Weng et al. (2018) studied predicting stock prices in a regression setting in the US stock market. Used machine learning methods included support vector regression and random forest regression. The used dataset combined technical indicators as well as data from news and google trends. As a result, random forest regression was the top performer with boosted regression trees.

Baser et al. (2023) predicted gold prices in regression setting using random forest, AdaBoost, XGBoost and gradient boost regressions. Daily market data was gathered between 2012 and 2019 and included 1685 observations. Simple and weighted moving

average, momentum, stochastic K%, stochastic D%, RSI, Williams R% and MACD were computed as technical indicators. Gradient boost regression performed the best in this study followed by XGBoost and random forest.

Picasso et al. (2019) combined technical and textual indicators to forecast stock market trends. Textual indicators included for example sentimentally analyzed news and technical indicators included for example RSI and simple moving average. The study was a binomial classification where target was to predict positive or negative trends for stocks. Models used for classification were random forest, support vector machine with radial kernel and feed-forward neural network. Dataset consisted of 20 largest market capitalization stocks in the nasdaq100 index, and for technical indicators the data was gathered in 15-minute intervals between 03.07.2017 – 14.06.2018. Feed-forward neural network was found to be the most robust of the models able to predict both positive and negative trends. Model was used for trading simulation where the models annualized return was 85,2% compared to 43,5% of the buy and hold benchmark.

Ayala et al. (2021) studied stock market index forecasting using machine learning. Authors studied four machine learning methods for regression, linear regression, feed-forward neural network with one hidden layer, random forest and support vector regression. Daily observations were gathered between start of 2011 and end of 2019 for IBEX, DAX and DJI. Technical indicators exponential moving averages and moving average convergence divergence were computed from the daily prices. In regression setting feed-forward neural network and liner regression performed the best, closely followed by random forest while random forest performed worst by a fair margin.

Worasucheep (2021) studied stock trading with ensemble classifiers. Considered classifies were support vector machine, random forest, feed-forward neural network and two gradient boost variants XGBoost and LightGBM. Daily stock data was gathered, and 45 technical indicators computed including RSI and MACD. After preprocessing with filtering, 11 indicators remained in the dataset. Stocks were classified to three classes,

high, low and medium which was the target of prediction. Author concludes that XBGoost and LightGBM outperform the other classifiers.

Mudassir et al. (2020) studied forecasting of bitcoin prices using machine learning methods. Considered machine learning methods were support vector machine, feed-forward artificial neural network, and neural network variants long-term short-term memory neural network as well as stacked artificial neural network. Prices were fore-casted for next day and the target was a binary classification of positive or negative returns. Predictor set consisted of technical indicators like RSI and simple, exponential and weighted moving averages. Three different intervals were used for prediction with varying results and stacked artificial neural network performed the best with average accuracy of 61,67 followed by support vector machine with average accuracy of 57,67. Feed-forward artificial neural network achieved average accuracy of 55,33 and long-term short-term memory artificial neural network achieved accuracy of 53,67.

Valencia et al. (2019) examined price prediction of the top four largest cryptocurrencies at the time, Bitcoin, Ethereum, Ripple and Litecoin. Considered machine learning methods were single-layer feed forward neural network, support vector machine and random forest. Hourly market data was collected for 60 days from February to April of 2018 and it was combined with sentiment analyzed tweets. Models predicted the price movement of next day in binary form, positive or negative movement. Predictions were based on three different scenarios, only market data, only sentiment analysis and both. Performance was measured with 5-fold cross-validation where single-layer feed-forward neural network performed the best with three of the four cryptocurrencies. The only exception was Litecoin where random forest performed the best using both sentiment and market data. For Bitcoin the results were as follows: neural network achieved accuracy of 0,39 on sentiment analysis only and 0,72 for marked data and both sentiment and market data. Support vector machine achieved accuracy of 0,5 for sentiment data, 0,55 for market data and both sentiment and market data. Random

forest had accuracy of 0,44 for sentiment as well as both market and sentiment data, and 0,61 for only market data.

3 Methodology

This section introduces the machine learning methods used in this study.

3.1 Machine learning methods

James et al. (2021) divide machine learning into two categories, supervised and unsupervised learning. In supervised learning an output is estimated based on one or more inputs. The output could be an exact amount or a class. With unsupervised learning there is still inputs but no outputs to supervise the learning with. Nevertheless, relationships and structures within the data can be learned in unsupervised learning. Machine learning used in this study is supervised learning and we are predicting the classes of observations.

Machine learning requires data from which to learn from. The whole dataset is divided to training and test datasets where training dataset is used to train the model and test dataset to measure models performance. In training the model is fed both the observations in the dataset as well as the correct labels. The model then tries to learn properties that observations with same class share. After training the model makes predictions on the test dataset. Model performance is measured by accuracy and in classification setting accuracy measure is the error rate of our model. Error rate is the proportion of incorrectly classified samples

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i), \tag{1}$$

where $\hat{y}_i$ is the predicted class label for observation i using our model. Indicator variable $I(y_i \neq \hat{y}_i)$ takes on value 1 if the prediction matches the true label, and 0 if the prediction and true label do not match.

Error rate is used to determine which model is the best. However, low error rate with training dataset does not guarantee a low error rate with test dataset. Often the best model in terms of training error rate has learned the properties and relationships of the training data which are not present in both training and test datasets. Therefore highly complex models can learn dataset specific properties and low error rate with training dataset will not reflect to low error rate with test dataset. This is called overfitting (see Figure 1). Instead the goal of machine learning is to find the model which finds the best generalization of the whole dataset and therefore has the lowest error rate in the test dataset.

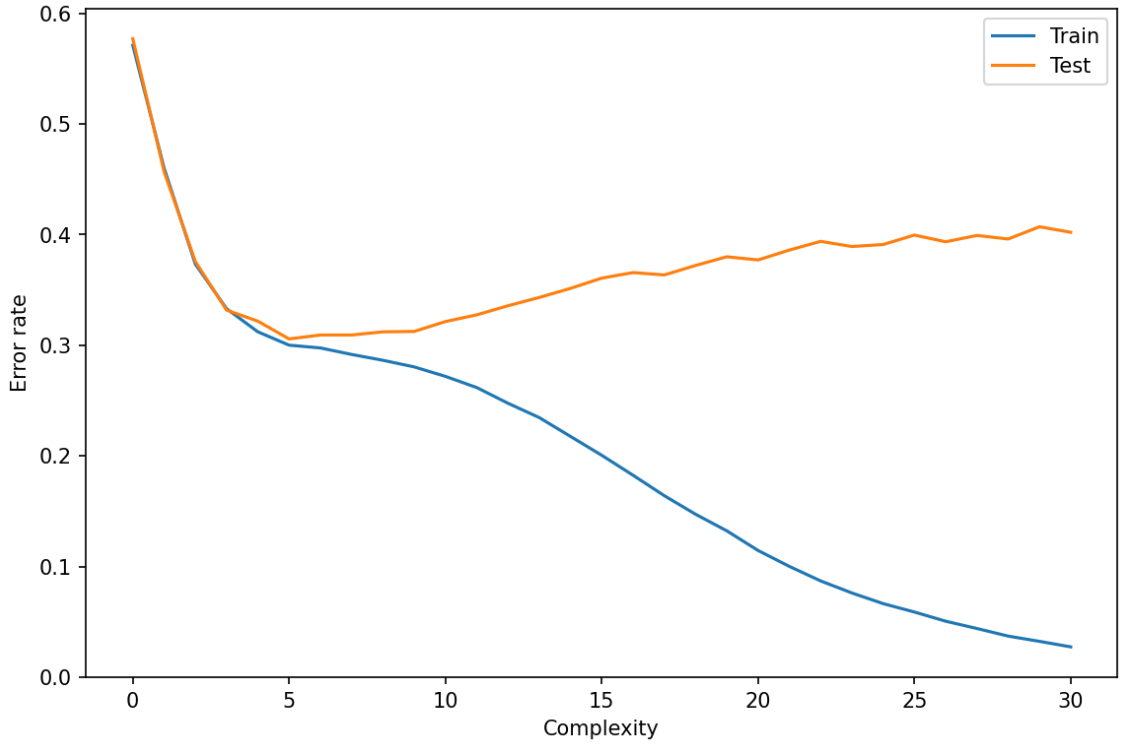

Figure 1. Overfitting. At first the error rate of a model declines for both training and test dataset when model complexity grows. However at some point the error rates deconverge and training error rate keeps on declining while the test error rate even starts to increase.

Machine learning methods used in this study are *k*-nearest neighbors classifier, gradient boosting classifier, random forest classifier, support vector machine classifier and artificial neural network. Each of these models has tweakable parameters which affect the model complexity, known as hyperparameters. To determine optimal hyperparameter values *k*-fold cross-validation, presented in section 4, is used. Method

implementation is done in Python where Keras library is used for artificial neural network and Scikit-Learn library is used for the other classifiers.

3.2 *k*-Nearest neighbors classifier

In *k*-nearest neighbor classifier, classification is based on existing datapoints of the dataset. Distance between new observation and existing datapoints is calculated and the closest *k* datapoints, nearest neighbors, are selected. The class of a new observation is determined by majority vote where the new observation is assigned to the most prominent class in the selected neighbors. If there is no single most common class, the assignment is determined randomly between tied classes. This process is repeated for all observations in the test dataset to determine their class. The most often used distance measure is Euclidean distance. *k*-value selection is important as small *k* is sensitive to mislabeled and noisy datapoints. On the other hand, a large value of *k* can include outliers from other classes (Imandoust & Bolandraftar, 2013). In this study cross-validation is used to determine the best value of *k*. Mathematically *k*-nearest neighbors are expressed as

$$\mathrm{PR}(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in N_0} I(\mathrm{y}_i = j) \tag{2}$$

where *k* is a positive integer, x_0 is a test observation and N_0 are the *k* nearest training observations (James et al., 2021, p. 39). Therefore *k*-nearest neighbors classifier calculates the conditional probability for class *j* as the fraction of points in N_0 that belong to class *j*. Test observation is classified to the class with highest conditional probability. Decision boundary of a *k*-nearest neighbors classifier and the decision criteria for new observations are shown in Figure 2.

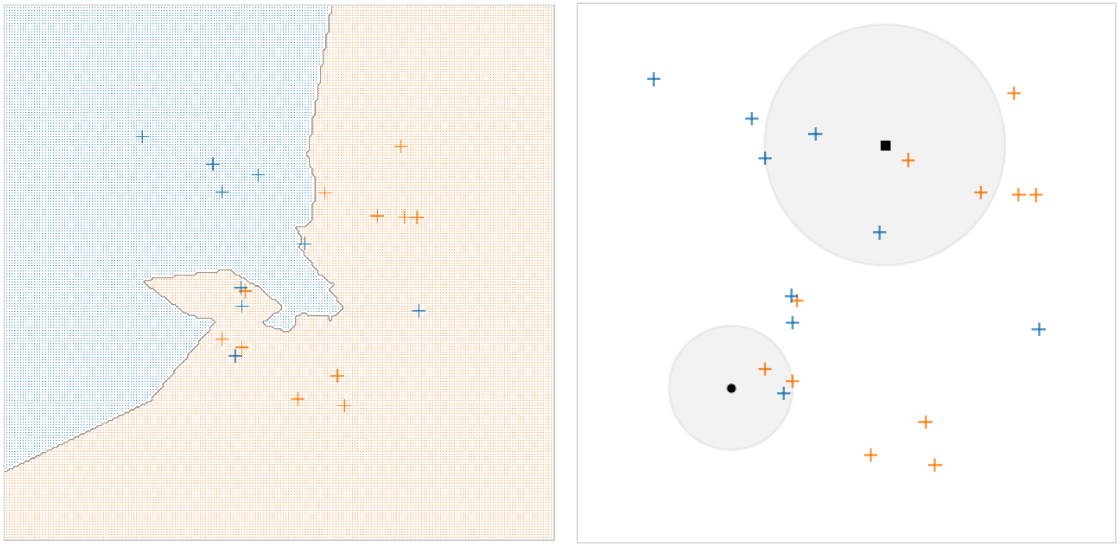

Figure 2. *k*-nearest neighbors classifier in two dimensional space. Left figure shows the decision boundary of a sample with *k*=5. Right figure shows the the decision criteria for two new observations. Observation with square symbol would be classified as blue when *k*=5 and observation with circle symbol would be classified as orange when *k*=3. Figures plot the same dataset and left figure is zoomed out to better show the decision boundary.

3.3 Random forest classifier

Random forest introduced by Breiman (2001) is an ensemble method which combines multiple simple models to a single model. The simple models are known as weak learners, as they lead to less accurate predictions on their own. Random forest uses decision tree classifier as the weak learner.

Decision trees are built by splitting the observations based on their features into J distinct and non-overlapping regions R_j (See Figure 3). Observations are classified to the most prominent class of their respective region. Mathematically a decision tree with J regions is written

$$f\left(x_{t-1};\{c_j,R_j\}_{j=1}^{J}\right)=\sum_{j=1}^{J}c_jI\left(x_{t-1}\in R_j\right), \tag{3}$$

where $c_j \in \mathbb{R}$ is the functional estimate in region R_j (Nevasalmi, 2020).

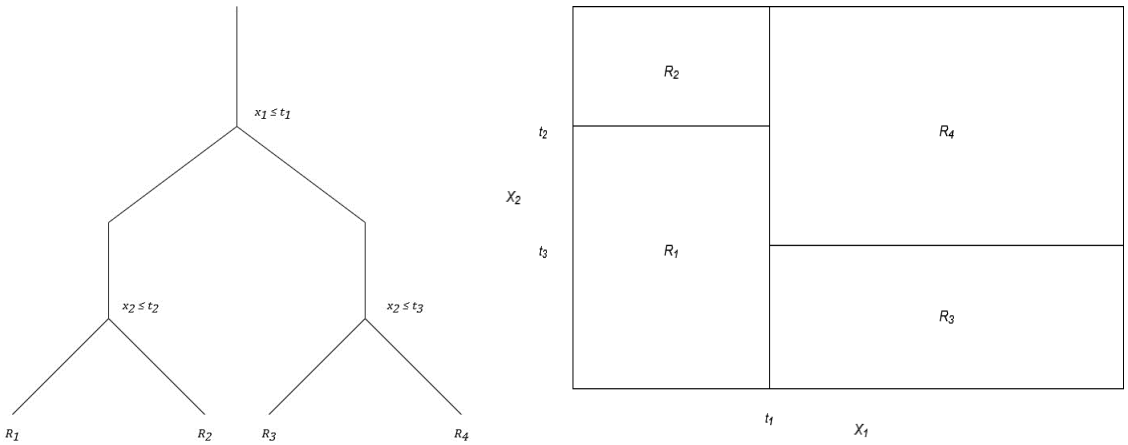

Figure 3. 4-node decision tree and the corresponding classification space (Nevasalmi, 2020).

A top-down greedy approach known as recursive binary splitting begins at the top of the three, where all observations belong to a single region. This single region is split into two regions based on decision criteria. Recursive binary splitting chooses the best possible decision criteria for the split by going through all predictors X_j and selecting the cut point s which splits the observations to two classes $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ the best. In this study we determine the best split with node purity using Gini index as a measure. Gini index is mathematically written as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{4}$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the *m*th region that are from the *k*th class (James et al., 2021, p. 336). Gini index takes on small values when all of the $\hat{p}_{mk}$ are close to zero or one and therefore a small value indicates that a region or node is pure and contains mostly observations from a single class (James et al., 2021, p 336). Recursive binary splitting continues to split the regions until all observations in a region are from a single class or a stopping criterion is reached. Stopping criterion could be for example the maximum depth of a tree, the minimum purity in-

crease of a split or the maximum amount of observations in a region. Recursive binary splitting could produce good results with the training data, but it has a high change of overfitting, resulting in poor performance with the test data because the tree is too complex. Furthermore, recursive binary splitting does not account for sequential splits where the first split would not increase node purity much, but the split of the resulting node would yield pure nodes.

Ensemble method bootstrap aggregation, also known as bagging, counters decision trees tendency to overfit using the bootstrap. In bootstrap new datasets are created from the original dataset by randomly drawing samples from the original dataset. Sampling is performed with replacement, a single sample from original dataset may appear more than once on the new dataset. In bagging B datasets are drawn, and a decision tree is built to each of the datasets to get prediction $\hat{f}^{*b}(x)$. Bagged predictions are then averaged (James et al., 2021, p. 341)

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)\,. \tag{5}$$

All decision trees are grown deep and as a result, they have high variance and low bias independently. However, averaging the predictions lowers the variance when hundreds or thousands of bagged datasets are drawn and corresponding decision trees are built. In classification the final prediction of the model is determined by majority vote. For a single observation the final predicted class is the class that was most often predicted during bagging.

Random forests are similar to bagged trees. Multiple decision trees are built on bootstrapped training datasets, but each time a new split is considered only a randomly drawn subset of predictors *m* are considered for the split. Considering only a subset of predictors decorrelates the resulting trees as the best variables are not always used for the split. Like with bagging, the final prediction is determined by majority vote and the

prediction for an observation is the class which received most predictions. In this study the value of *m*, number of bagged trees and the minimum number of observations in a node are determined with cross-validation. Random forests are presented in Algorithm 1.

Given training set $\{(x_i, y_i)\}_{i=1}^{n}$, and number of iterations B

    for $b = 1$ to B:
        draw a bootstrap sample Z of size N from training set
        while (number of observations in some node > n_{min}):
            randomly select m variables from p
            split the current node with the best split point
        end while
        add decision tree $f_b(x_{t-1})$ into the ensemble F_b
    end for

Algorithm 1. Random forest Algorithm (Nevasalmi, 2020).

3.4 Gradient boosting classifier

Gradient boosting algorithm, which can be used for both classification and regression, was introduced by Friedman (2001). Gradient boosting for classification builds an ensemble model which allows for the optimization of arbitrary differentiable loss functions. In each stage decision trees are fit on the negative gradient of the loss function, so called pseudo residuals. Therefore, gradient boosting seeks to minimize the expected loss of some loss function. In binary classification problem $y_t \in \{0,1\}$ the loss function is the binomial deviance

$$L_{\log(y,p)} = -(y \log(p) + (1-y) \log(1-p)) \quad (6)$$

where $p = \Pr(y = 1)$ (Scikit-learn). Binomial deviance can be extended to multiclass case as multinomial deviance which is used in this study. These loss functions can be used to evaluate the probability outputs of a classifier.

Gradient boosting classifier is trained sequentially, and decision trees are fit on the residuals of the earlier model. This way the model tries to correct the mistakes done in the previous iteration of the model.

Gradient boosting is an ensemble method like Random Forest. In ensemble machine learning models a prediction is built by combining the strengths of simple base models. The base model for our algorithm is a decision tree. In random forest classifier decision trees were built deep, but in gradient boosting the tree depth is limited. Ensemble method can be presented as

$$f_M(x) = \sum_{m=1}^{M} f_m(x_{t-1}), \tag{7}$$

where M is the number of base models and the final prediction of observation x is given as a sum of base model predictions (Nevasalmi, 2020). Gradient boosting is represented algorithmically in Algorithm 2.

1. Initialize with a constant value:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$

2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute:

$$r_{im} = -\left[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\right]_{f=f_{m-1}}$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1,2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$y_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

   (d) Update

$$f_m(x) = f_{m-1}(x) + \delta \sum_{j=1}^{J_m} \gamma_{jm} I\big(x \in R_{jm}\big).$$

3. Output $\hat{f}(x) = f_M(x)$.

Algorithm 2 Gradient Tree Boosting Algorithm (Hastie et al., 2017, p. 361).

Given a differentiable loss function $L(y_i, y)$ we initialize the model with a constant value of $f_o(x)$ which minimizes the loss function. *M* is the number of iterations performed in gradient boosting and for each iteration *m* a pseudo residual or target r_{im} is calculated for all observations. A decision tree is fit to the data based on the pseudo residuals r_{im}. This differs from random forest where the class y_i is used to fit the model. The output value of each terminal region y_{jm} in the fitted tree is calculated. New prediction for each sample $f_m(x)$ is computed using the prediction from previous iteration $f_{m-1}(x)$, the learning rate δ and the output values of the new tree γ_{jm}. The output of gradient boosting model for each observation $\hat{f}(x)$ is the sum of original prediction from the first decision tree and the following predictions scaled by learning rate.

In this study the learning rate is 0.1 and the parameters for number of iterations and maximum depth of the decision trees are determined by cross-validation. In addition, the share of predictors used for each split and the share of training samples used for each base learner are set by cross-validation.

3.5 Support vector machine

Support vector machines were introduced by Cortes and Vapnik (1995). In a two-class classification suppose that training data consists of two dimensions and there exists a hyperplane which can perfectly separate the two classes. There are multiple possible hyperplanes which will separate the data correctly, but the optimal hyperplane is the maximal margin hyperplane. Margin is the perpendicular distance between hyperplane and the closest training observation and maximal margin hyperplane has the largest margin. Maximal margin hyperplane is the solution to following optimization problem

$$\underset{\beta_0, \beta_1, \ldots, \beta_p, M}{\text{maximize } M}$$

$$\text{Subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \tag{8}$$

$$y_i\left(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_i X_{ip}\right) \geq \forall\, i = 1, \ldots, n$$

where M is margin (James et al. 2021, p. 372). With maximal margin classifier we can calculate the hyperplane and classify new observations based on to which side of the hyperplane they belong (See Figure 4). Also, if the new observation is far away from the hyperplane the correct classification has a larger probability. However, maximal margin classifier works only If there exists a hyperplane which perfectly separates the classes.

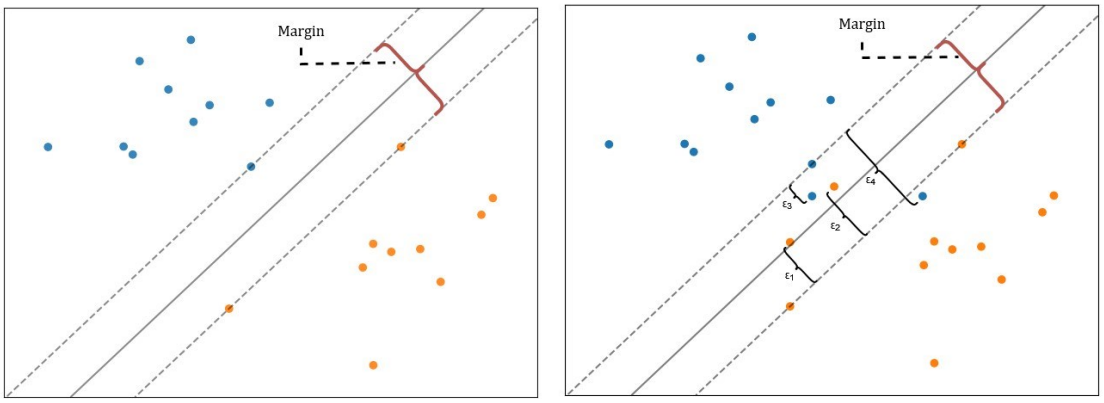

Figure 4. Maximal margin hyperplane. Left figure shows maximal margin hyperplane and the margin in a linearly separable case. In right figure the data is not linearly separable, but by allowing margin and hyperplane to be violated with slack variables a hyperplane is found.

Support vector classifier introduces a soft margin to maximal margin classifier. Soft margin allows part of the observations to be incorrectly classified in order to improve model robustness to individual observations and to better classify most of the observations. With support vector classifier it is possible for observations to be on the wrong side of the soft margin and even on the wrong side of the hyperplane. The optimal hyperplane for support vector classifier is the solution to the following optimization problem

$$\begin{aligned} &\underset{\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots, \epsilon_p M}{\text{maximize } M} \\ &\text{Subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \\ &y_i \left(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_i X_{ip} \geq M(1 - \epsilon_i),\right. \\ &\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \end{aligned} \tag{9}$$

where M is the margin and $\epsilon_i, \ldots, \epsilon_n$ are called slack variables which allow observations to be on the wrong side of the margin or hyperplane (James et al. 2021, p. 375). The slack variable tells observations position in regard to the hyperplane. If $\epsilon = 0$ the observation is on the correct side of margin, if $\epsilon > 0$ the observation is on the wrong side of margin and if $\epsilon > 1$ the observation is on the wrong side of the hyperplane. In the optimization problem C is a positive tuning parameter. C is the sum of slack variables and controls how much margin and hyperplane violations are allowed in the classification. A larger value of C allows more violations of margin and hyperplane, and conversely a smaller value allows less violations. In this study the value of C is determined with cross-validation.

Only observations which lie either on the margin or on the wrong side of it affect the classifier and these observations are known as support vectors. C controls the generalization capability of a model. When C is large, there are multiple support vectors, and the model is not so sensitive to single observations and when is C small there are fewer support vectors, and the model is more sensitive to single observations. Support vector classifier works well when the data is linearly separable but struggles with nonlinear data. The support vector machine solves this by enlarging the feature space using kernels.

The solution to optimization problem in equation 9 involves only the inner products of the observations, given by

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij}\, x_{i'j} \tag{10}$$

and the linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle \tag{11}$$

where there are n parameters $\alpha_i, i = 1, \ldots, n,$ one per training observation (James et al. 2021, p. 381). To estimate the parameters α_i and β_0 the inner products between all pairs of observations is needed. The parameter α_i is nonzero only for observations which are support vectors.

It is possible to replace the inner products in above equations 10 and 11 with a generalization of the inner product of form $K(x_i, x_{i'})$ where K is know as the Kernel function. Typical kernel functions are polynomial and radial kernels.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j}\right)^d . \tag{12}$$

Equation 12 describes the polynomial kernel of degree d, where d is a positive integer (James et al., 2021, p. 382). In this study we use polynomial kernels of second and third degree. Radial kernel is described as

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} x_{i'j})^2\right) \tag{13}$$

where γ is a positive constant (James et al., 2021, p. 382). γ is determined with cross validation in this study. Radial kernel behaves similarly to weighted *k*-nearest neighbors model, where the closest observations have the most say on individual observations class. Support vector machines with polynomial and radial kernels are displayed in Figure 5.

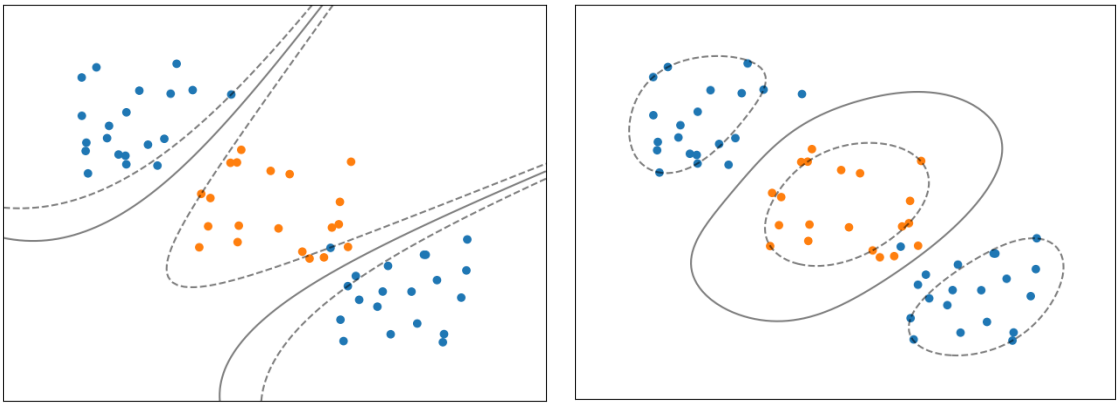

Figure 5. Support vector machine. Left figure shows support vector machine with 3rd-degree polynomial kernel fitted to data and right figure shows radial kernel fit to data. Both models are able to separate the classes.

For *K*-class classification a one-versus-one approach is used where $\binom{K}{2}$ classifiers are constructed. Each classifier compares two classes and in the end the observation is assigned to the class which has the most assignments overall.

3.6 Artificial neural network

The model used in this study is a single hidden layer feed-forward neural network which consists of three layers, input layer, hidden layer and output layer. The network is visualized in Figure 6. Each layer is connected to next with weights which connect each node in the previous layer to each node in the next layer. Neural networks can be built with more than one hidden layer to capture more complex relationships between variables, but this can lead to overfitting. The amount of input layer nodes equals the

amount of variables in the dataset while number of nodes in hidden layer can be determined. In our model the number of nodes in hidden layer is determined with hyperparameter tuning.

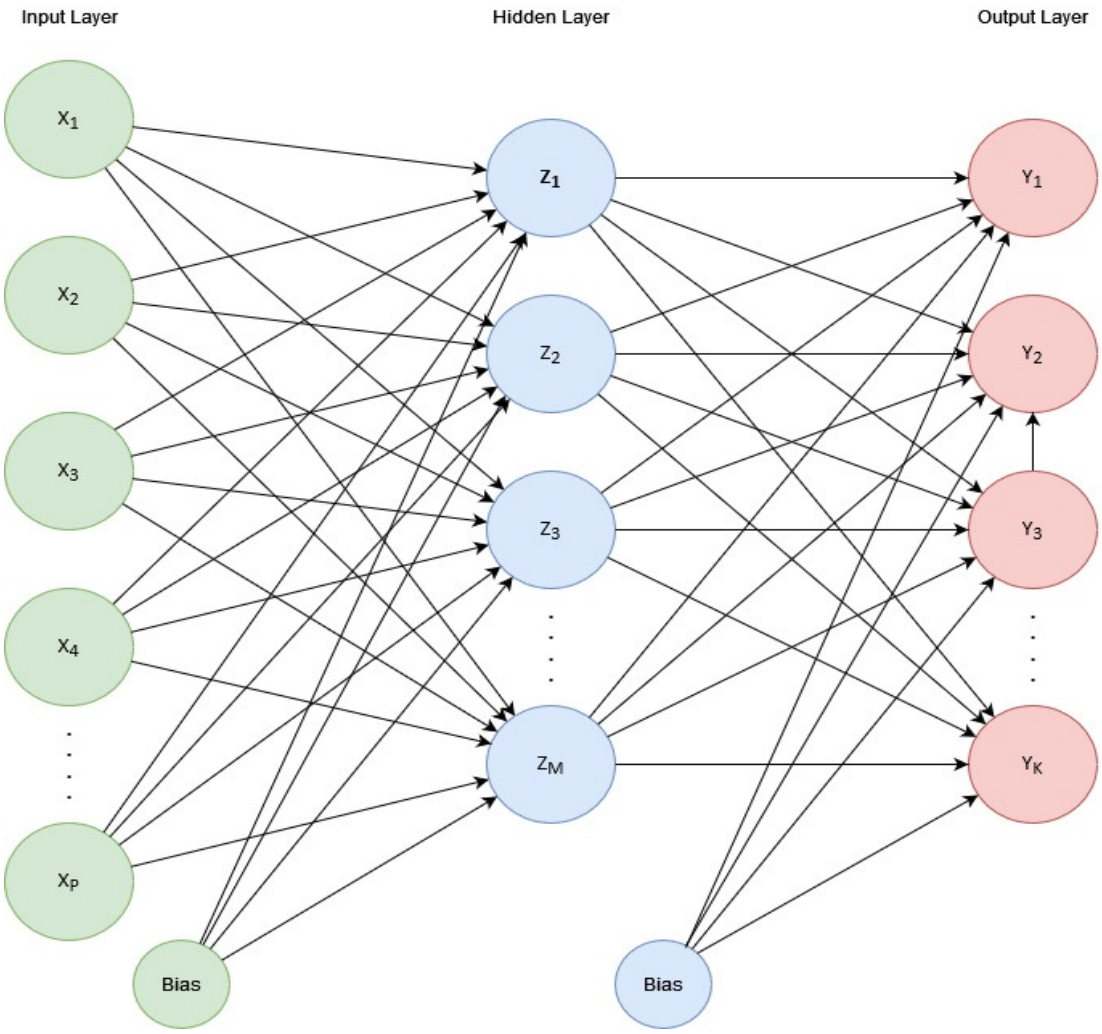

Figure 6. Feed-forward neural network.

Hidden layer nodes are linear combinations of the input variables X = (X_1, X_2, …, X_P) transformed by an activation function

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \dots, M, \quad (14)$$

where α_{0m} is the bias, α_m^T is a vector of weights connecting input layer units to output layer units and $\sigma(x)$ is the activation function (Nevasalmi, 2020). Activation function transforms the model from linear to non-linear. Usual selections for activation function are rectified linear unit or the sigmoid function which are visualized in Figure 7.

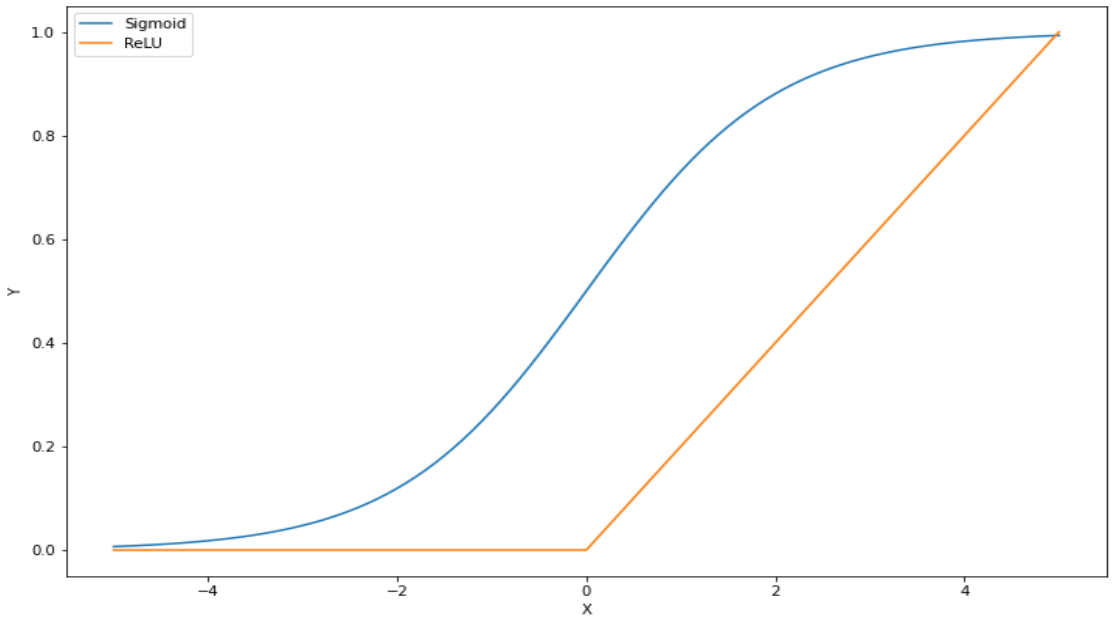

Figure 7. Sigmoid and ReLU activation functions.

In this study the hidden layer uses rectified linear unit as the activation function. Rectified Linear Unit (ReLU) is described in equation 15.

$$f(x) = \max(0, x) = \begin{cases} 0, & if\ x \leq 0 \\ x, & if\ x > 0 \end{cases} \tag{15}$$

Sigmoid function is described in equation 16.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{16}$$

The final output of the model is a linear combination of the hidden units Z

$$f_k(X) = g(\beta_{0k} + \beta_K^T Z), k = 1, \dots, K, \quad (17)$$

where β_{0k} is the bias and β_K^T is the vector of weights connecting hidden layer units to output layer unit (Nevasalmi, 2020). The output layer uses activation function SoftMax, described in equation 18, which transforms output values to sum to 1. Therefore, the values represent probabilities of sample belonging to respective class.

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \quad (18)$$

Optimal weights in a neural network minimize the selected loss function. For multiclass classification problem the loss function is categorical cross-entropy

$$L(\theta, F_k) = -\sum_{t=1}^{N}\sum_{k=1}^{K} I(R_t = k) \log F_k\,, \quad (19)$$

where θ is a weight vector denoting the complete set of weights in the network and F_k is the output for class k (Nevasalmi, 2020). The weights of the network are optimized using backpropagation technique. In backpropagation the gradient of the loss function regarding model weights is calculated and the weights are moved to opposite, negative direction of the gradient by a small amount. The negative gradient is multiplied by an amount called the learning rate and it is set to 0,01 in this study.

In practice the training dataset is divided into batches to lessen memory usage of computations. Training of neural network is described in algorithm 3. Once all the batches have been through training an epoch is completed. The number of epochs to train is 100 in this study.

```
for each batch:
     Compute the predictions of the model for samples in
     batch.
     Compute the loss value for these predictions.
     Compute the gradient of the loss regarding model's
     weights.
     Move the weights by a small amount in the opposite di-
     rection of gradient.
```

Algorithm 3. Artificial neural network (Chollet, 2021, p. 64-65).

There are multiple strategies to avoid overfitting with neural networks. One strategy is to use regularization called weight decay, where a penalty term is added to loss function to penalize large weights.

$$l(\theta, F_k) = L(\theta, F_k) + \lambda J(\theta), \tag{20}$$

where λ is the weight decay parameter (Nevasalmi, 2020).

Another strategy to avoid overfitting is called early stopping. In early stopping a subset of samples are collected to so called validation dataset. These samples will not be used to train the model but to validate its performance. Early stopping monitors the loss of the validation set and once the loss stops improving, the training will be stopped. In this study the proportion of samples for validation dataset is 0,2. Patience parameter determines after how many epochs of no improvement of loss the training will be stopped, and in our study the parameter is set to 5. Early stopping is visualized in Figure 8. The loss of training continues to lower as more epochs are trained while loss of validation stops improving early on and starts to worsen. This indicates overfitting as the model finds relationships in the training data which do not appear in validation data.

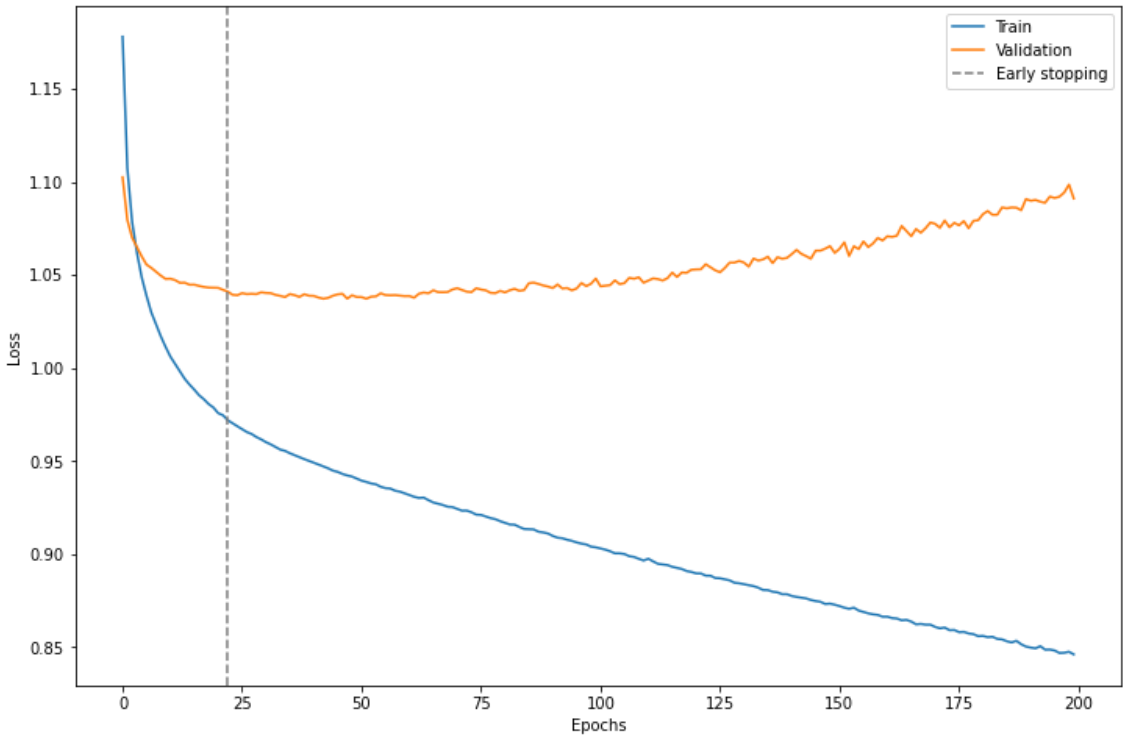

Figure 8. Early stopping. Dashed line shows the point where early stopping stops training.

4 Data

Dataset consists of 8014 daily observations ranging from 2.9.1991 to 19.5.2022. Dataset is split into training and test datasets where 75 percent of observations are in the training dataset and remaining observations in test set. Training data ranges from 2.9.1991 to 12.9.2014 while test data ranges from 15.9.2014 to 19.5.2022. Training dataset is used to train the models and validate their performance while test dataset is used to measure predictive performance.

Dependent variable in this study is next days daily returns transformed into a multinomial response variable following Nevasalmi (2020). Multinomial response variable with three classes is derived using two thresholds, c1 and c2.

$$R_t(c_1, c_2) = \begin{cases} 1, & \text{if } Z_t < c_1 \\ 2, & \text{if } c_1 \leq Z_t \leq c_2 \\ 3, & \text{if } Z_t > c_2 \end{cases} \quad (21)$$

The thresholds are determined to be upper and lower quartiles of the daily returns of OMXH25 index where the upper quartile is 0,795 percent return and the lower quartile -0,673 percent return. With multinomial response variable the large negative or positive returns of a market day are in focus compared to binary response variable where only the direction of the stock market is considered. For trading purposes, the larger daily price movements are more interesting as forecasting these correctly affects profits more than the close to zero price movement days even if their direction is correctly predicted.

The development of OMXH25 index price is presented in Figure 9. Of the 8014 daily returns 4056 are positive, 3625 are negative and 333 are zero. Therefore, the dataset is well balanced.

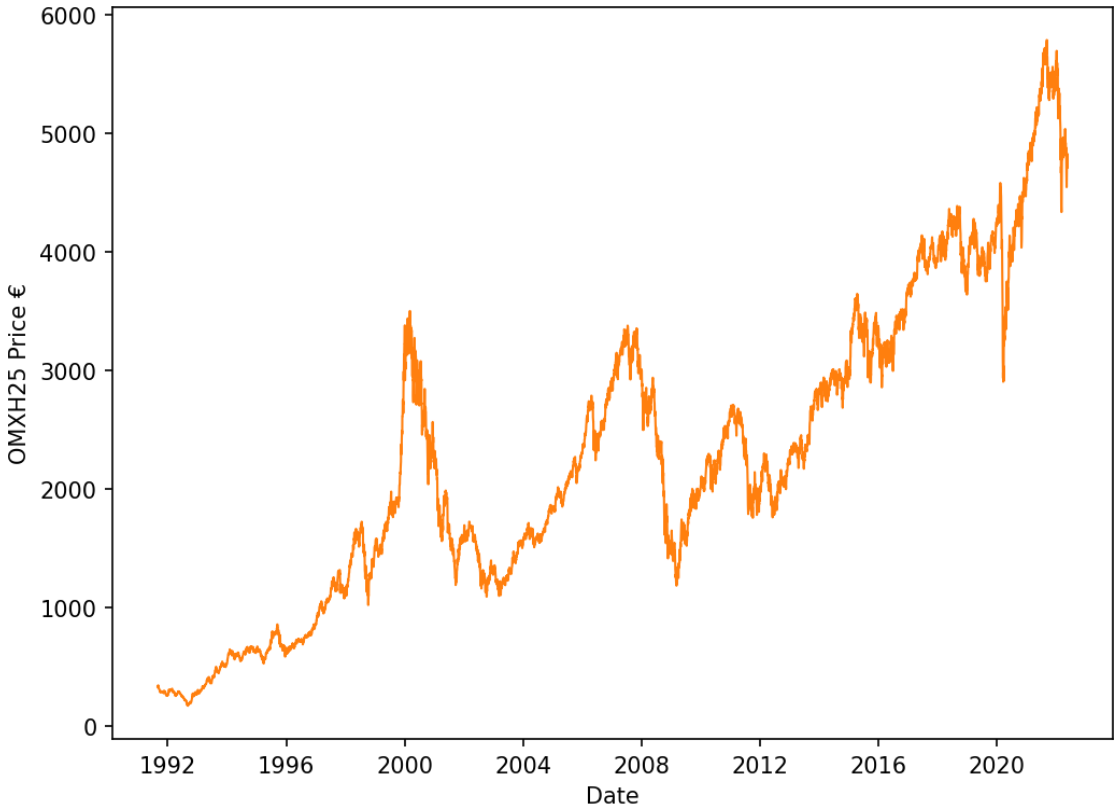

Figure 9. Price development of OMXH25.

Independent variables include data from stock markets, commodity market and bond market. Stock market data includes daily returns of major stock indices SP500, FTSE100 and DAX. Missing values due to national holidays are filled using forward fill and the value of previous day with data is used. Additionally, multiple technical indicators are calculated from OMXH25 daily. Technical indicators include moving average, momentum, stochastic K% and D%, relative strength index, Williams R% (LWR), moving average convergence divergence and Aroon oscillator. Standard deviation of last 251 days is used as volatility indicator.

Commodity variables are the daily return of gold, silver, copper and oil. For interest rates 10-year Finnish bond rate and 3-month FIBOR rate are considered. Additionally, term spread is calculated as difference of the 10-year and 3-month rates. Calculation formulas for indicators are presented in Table 1.

Table 1. Indicator formulas. For all indicators n = 10. H_n is the highest price of previous *n* days whereas L_n is the lowest price. positives is the sum of positive daily returns and negatives is the sum of negative daily returns in the previous *n* days. ewm is the exponential weighted moving average of previous 12 and 26 days. Aroon up and Aroon down are specified for simplicity.

Indicator	Formula
Stock market	
OMXH25, SP500, FTSE100, DAX,	$\frac{P_t}{P_{t-1}} - 1$
Commodities	
Gold, Oil, Copper & Silver	$\frac{P_t}{P_{t-1}} - 1$
Interest rates	
10 year Finnish bond & 3 month FIBOR	$P_t - P_{t-1}$
Term spread	$10\ year\ Finnish\ bond - 3\ month\ fibor$
Technical analysis	
Moving average	$n^{-1} \sum_{i=0}^{n-1} P_{OMXH25,t-1}$
Momentum	$P_{OMXH25,t} - P_{OMXH25,t-(n-1)}$
Stochastic K%	$k_t = \frac{P_t - L_n}{H_n - L_n} \times 100$
Stochastic D%	$n^{-1} \sum_{i=0}^{n-1} k_{t-1}$
RSI	$\frac{positives}{positives + negatives} \times 100$
LWR	$\frac{H_n - P_t}{H_n - L_n} \times 100$
MACD	$macd_t = ewm_{t-12} - ewm_{t-26}$
Aroon up	$100 \times \frac{(25 - periods\ since\ 25 - period\ high)}{25}$
Aroon down	$100 \times \frac{(25 - periods\ since\ 25 - period\ low)}{25}$
Aroon oscillator	$Aroon\ up - Aroon\ down$

Lag variables up to 10 days are considered for stock market variables. Summary statistics of variables are presented in Table 2. All variables are scaled to have zero mean and unit variance.

Table 2. Summary statistics.

Variable	Mean	Standard deviation	Minimum	Lower Quartile	Median	Upper Quartile	Max
Stock market							
OMXH25	0,0004	0,0145	-0,1013	-0,0067	0,0002	0,0079	0,0973
SP500	0,0003	0,0113	-0,1198	-0,0041	0,0003	0,0054	0,1158
FTSE100	0,0002	0,0110	-0,1087	-0,0050	0,0001	0,0057	0,0984
DAX	0,0004	0,0138	-0,1224	-0,0058	0,0005	0,0071	0,1140
Commodities							
Gold	0,0003	0,0098	-0,0849	-0,0042	0,0001	0,0049	0,1100
Oil	0,0006	0,0268	-0,6143	-0,0097	0,0000	0,0111	0,7740
Copper	0,0003	0,0150	-0,0986	-0,0070	0,0000	0,0075	0,1264
Silver	0,0004	0,0180	-0,1771	-0,0072	0,0000	0,0089	0,1328
Interest rates							
10 year Finnish bond	-0,0012	0,0532	-0,8900	-0,0274	-0,0004	0,0210	0,7520
3 month FIBOR	-0,0012	0,0178	-0,3580	-0,0026	0,0000	0,0020	0,3860
Term spread	1,47	1,18	-1,32	0,70	1,27	2,02	6,38
Volatility							
Standard deviation of previous 251 days	184,86	118,52	24,76	104,45	159,70	227,03	639,53
Technical analysis							
Momentum 10	5,44	106,52	-1192,75	-34,98	10,90	57,57	609,28
Moving average 10	2267,91	1285,73	181,76	1307,34	2118,74	3143,68	5728,88
Stochastic K%	56,84	38,52	0,00	18,34	64,14	98,94	100,00
Stochastic D%	56,84	28,26	0,00	33,23	59,77	81,94	100,00
RSI	53,97	20,93	0,00	38,67	54,17	69,83	100,00
LWR	-43,16	38,52	-100,00	-81,66	-35,86	-1,06	0,00
MACD	3,91	37,91	-365,25	-9,66	7,37	24,68	221,76
Aroon Oscillator	14,14	62,07	-96,00	-48,00	36,00	68,00	96,00

Gradient boost and random forest classifiers calculate the relative importance of predictors. The relative importance of predictors with both classifiers are summed and the ten most influential predictors are presented in Table 3.

Table 3. Variable importance.

Rank	Description	RF importance	GB importance	Sum of importance
1	SP500	0,0584	0,4024	0,4608
2	Momentum 10	0,0179	0,0695	0,0874
3	MACD	0,0182	0,0488	0,0669
4	OMXH25	0,0218	0,0388	0,0605
5	Moving average 10	0,0170	0,0418	0,0588
6	OMXH25 2 day lag	0,0183	0,0238	0,0420
7	OMXH25 5 day lag	0,0181	0,0224	0,0405
8	DAX 1 day lag	0,0168	0,0216	0,0384
9	3 month FIBOR	0,0191	0,0177	0,0368
10	OMXH25 1 day lag	0,0168	0,0132	0,0300

The previous days return of SP500 is the most important predictor. Finnish stock market closes two hours after the US stock market opens, with some variety due to daylight savings time. The direction of SP500 with finnish stock market seems to correlate highly with the next days returns of OMXH25. Technical indicators rank highly in the most important variables as 10-day momentum and MACD rank second and third. In addition, 10 day moving average ranks fifth most important. OMXH25 lag variables of 1, 2, 3 and 6 as well as 2-day lag of DAX index rank among the ten most important variables. Additionally, short-term bond rare 3-month FIBOR ranks as ninth most important.

Nevasalmi (2020) combined a reduced dataset from the ten most important features and trained *k*-nearest neighbors, support vector machine and feed-forward neural network with the reduced dataset to counter overfitting. Cross-validation results of the reduced and original dataset were compared, and the results did not differ significantly and therefore the whole predictor set was used in training.

5 Hyperparameter optimization

Test error rate can be estimated with validation set approach. In this approach the training set is randomly divided into two sets, a training set and a validation set. Model is trained with the training set and the validation set is used to measure models performance. James et al. (2021, p. 200) notice two issues with validation set approach. Firstly, the test error rate estimate from validation set approach can be highly variable. Observations chosen for the validation set are not used to train the model and therefore the model does not account for these observations. In the worst-case scenario, a fraction of observations share a property, but all of these observations are included in the validation set and thus the model trained with the training set cannot take this property into account. Additionally, validation set approach uses only a subset of observations for training which tends to lead to worse performance.

k-fold cross-validation counters the drawbacks of validation set approach. In *k*-fold cross-validation the training set is randomly divided into *k* sets or folds with equal number of observations. Model is trained with the first fold as a validation set and rest of the folds as training set to get validation error rate. This is repeated *k* times and each fold is used as a validation set in turn. Cross-validation error rate estimate CV is calculated as an average of the validation test errors

$$CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} error\ rate_i\,. \tag{22}$$

where usual values for *k* are 5 and 10 (James et al., 2021, p. 203). In this study we use the latter. *k*-fold cross-validation procedure is shown in Figure 10. *k*-fold cross-validation has a higher computational expense, but it provides valuable estimates of model performance.

k-fold cross-validation is used to determine the optimal hyperparameter values for

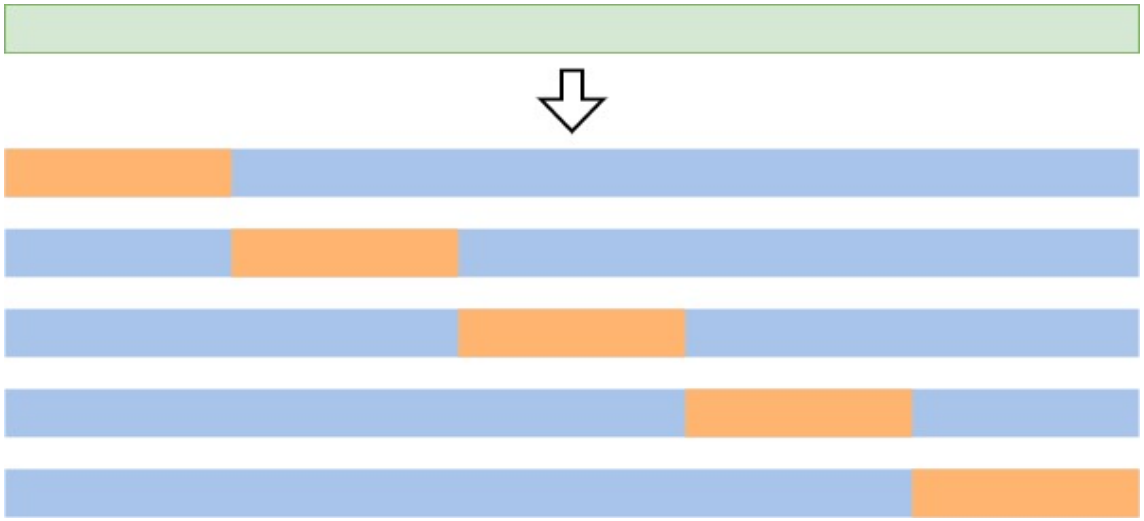

Figure 10. *k*-fold cross-validation with *k* = 5. Green bar presents the whole training set, which is then divided to 5 folds. Each fold is used as validation set, represented with orange color, where remaining folds are used for training, represented with blue color.

each method. Each methods hyperparameters and their considered values are presented in Table 4. Cross-validation error rate is computed for all considered hyperparameter value combinations and the values with best cross-validation error rate are chosen for the actual models used in this study.

Table 4. Considered hyperparameters and their values for each machine learning method.

Method	**Description**	**Notation**	**Considered values**
k-nearest neighbors	number of neighbors	k	11, 21, …, 461
Gradient boosting	Number of iterations	M	2, 3, 4, 5, 10, 15, 20, 25, 30, 50, 100, …, 500
	Tree depth	D	2, 3
	Fraction of training points	n_{row}	0.5, 0.7, 0.9
	Fraction of predictors	n_{col}	0.7, 0.9
Random forest	Number of trees	B	100, 300, 500
	Number of predictors	n_{row}	0.25, 0.5, 0.75
	Observations in each node	n_{col}	10, 50, 100, 200, …, 600
Support vector machine	Cost parameter	C	0.01, 0.1, 0.25, 0.5, 1, …, 9
	Radial kernel parameter	γ	0.005, 0.01
	Polynomial kernel, scal	s	0.005, 0.01
	Polynomial kernel, degree	d	2, 3
Artificial neural network	Number of hidden units	M	3, 5, 8, 10, 12, 15, 20, 30

Optimal hyperparameters based on cross-validation are presented in Table 5. These values are used to train the actual models used in this study.

Table 5. Hyperparameter optimization results.

Method	**Description**	**Notation**	**Selected value**
k-nearest neighbors	number of neighbors	k	71
Gradient boosting	Number of iterations	M	25
	Tree depth	D	2
	Fraction of training points	n_{row}	0.5
	Fraction of predictors	n_{col}	0.9
Random forest	Number of trees	B	300
	Number of predictors	n_{row}	0.75
	Observations in each node	n_{col}	200
Support vector machine	Cost parameter	C	5
	Radial kernel parameter	γ	0.01
Artificial neural network	Number of hidden units	M	30

6 Empirical results

6.1 Statistical predictive performance

Statistical predictive performance is measured with accuracy, which is the share of correctly classified observations in the test dataset. Predictive performance of models is presented in table 6. Validation column includes the results of *k*-fold cross-validation while Train and Test column includes the results of models with train and test datasets respectively.

Artificial neural network achieves the best cross-validation results with 52,4 while rest of the models are around 47 and 48. Prediction on train dataset is below 50 percent for *k*-nearest neighbors and support vector machine models while random forest model has the best result with 55,4 percent. Random forest had the worst cross-validation results but is the best model based on training dataset. Results on test set are centered around 0,56 and 0,57 with artificial neural network being the outlier with 0,54. The best machine learning method in terms of statistical predictive performance is random forest classifier with accuracy of 0,5684. Performance improves with test data compared to training data for all models.

Table 6. Prediction results.

Method	Validation	Train	Test
k-nearest neighbors	0,4800	0,4940	0,5664
Gradient boost	0,4757	0,5283	0,5634
Random forest	0,4696	0,5541	0,5684
Support vector machine	0,4852	0,4712	0,5664
Artificial neural network	0,5241	0,5374	0,5434

Predictions are presented in confusion matrix format where the actual classes are in rows and predicted classes in columns. Correct predictions are on the diagonal where both predicted and actual class match. Confusion matrix for *k*-nearest neighbors model is presented in Table 7. The model predicted 38 out of total 436 low return days cor-

rectly. For high return days 16 out of 440 were classified correctly. For the center class 1081 out of 1128 days were classified correctly. This ratio is the true positive rate of prediction and it was 0,09 for class 1, 0,96 for class 2 and 0,04 for class 3. Precision is the ratio between the correctly classified samples and the total samples classified to a class. Precision of *k*-nearest neighbors model for class 1 was 0,34 and 0,43 for class 3 while class 2 had precision of 0,58.

Table 7. *k*-nearest neighbors confusion matrix.

		Predicted		
		1	2	3
Actual	1	38	384	14
	2	40	1081	7
	3	34	390	16

The results of Gradient boosting classifier are presented in Table 8. Gradient boosting classifier achieved true positive rate of 0,24 for class 1, 0,86 for class 2 and 0,14 for class 3. Achieved precisions were 0,4, 0,6 and 0,4 for classes 1, 2 and three respectively. It is notable that gradient boosting classifier predicted more of classes 1 and 3 compared to *k*-nearest neighbors classifier.

Table 8. Gradient boosting classifier confusion matrix.

		Predicted		
		1	2	3
Actual	1	103	304	29
	2	102	965	61
	3	51	328	61

The results of random forest classifier are presented in Table 9. True positive rates were 0,22 for class 1, 0,86 for class 2 and 0,18 for class 3. Precisions were 0,39, 0,61 and 0,43 for classes 1, 2 and 3 respectively. Like gradient boosting classifier, random forest classifier predicted more of classes 1 and 3.

Table 9. Random forest classifier confusion matrix.

		Predicted		
		1	2	3
Actual	1	96	306	34
	2	95	966	67
	3	58	305	77

Support vector machine, presented in Table 10, predicted less of classes 1 and 3 compared to gradient boosting and random forest classifiers, but more than *k*-nearest neighbor classifier. Support vector machine achieved true positive rate of 0,16 for class 1, 0,92 for class 2 and 0,09 for class 3. Precisions were 0,43, 0,59 and 0,46 for classes 1, 2 and 3 respectively.

Table 10. Support vector machine confusion matrix.

		Predicted		
		1	2	3
Actual	1	70	346	20
	2	61	1042	25
	3	33	369	38

Artificial neural network, presented in Table 11, achieved true positive rates of 0,33, 0,8 and 0,09 and precisions of 0,33, 0,61 and 0,44 for classes 1, 2 and 3 respectively. It is notable that artificial neural network predicted class 1 a lot compared to class 3.

Table 11. Artificial neural network confusion matrix.

		Predicted		
		1	2	3
Actual	1	143	274	19
	2	191	907	30
	3	98	303	39

All models predicted class 1 more than class 3, but the most striking difference was in the artificial neural network model which predicted a lot of class 1 and less of class 3. *k*-nearest neighbors model was the most conservative in its predictions, giving the least predictions to classes 1 and 3. On the other hand, gradient boosting classifier, random forest classifier and artificial neural network were more active in predicting classes 1 and 3.

6.2 Economical predictive performance

In addition to statistical predictive performance, also the economical predictive performance is measured. This is achieved by creating a trading strategy based on models predictions. Trading strategy is the same as in Nevasalmis (2020) study and is presented in Table 12. This trading strategy depends on current ($\hat{R}_{t+1}$) and previous ($\hat{R}_t$) predictions. Only full and zero allocation to stock market are considered. To determine if a model is in the market or not is based on trading strategy and model predictions.

Table 12. Trading strategy.

		$\hat{R}_t$		
		1	2	3
$\hat{R}_{t+1}$	1	Stay out	Sell	Sell
	2	Buy	Hold	Hold
	3	Buy	Hold	Hold

Transaction costs are taken into account similarly to Nevasalmis (2020) study. When model simulations status changes from being in the market to not being in the market or vice versa 0,1 percent transaction costs are subtracted.

Results of trading are presented in Table 13. Buy and hold strategy, being in the market for the whole period, is considered as benchmark. Benchmark returned 160,3 percent while the best performing model, gradient boost, returned 355,98 percent. Compared to benchmark gradient boost model performed 222,07 percent better. Support vector machine model performed nearly as well as gradient boost returning 355,05 percent. *k*-nearest neighbors performed worst returning 188,6, while random forest and artificial neural network returned 278,94 and 276,1 respectively. Model performance is visualized in Figure 11.

Table 13. Trading simulation results.

Method	% Return	% compared to benchmark
Buy & Hold	160,30	-
k-nearest neighbors	188,60	117,65
Artificial neural network	276,10	172,24
Random forest	278,94	174,01
Support vector machine	355,05	221,49
Gradient boost	355,98	222,07

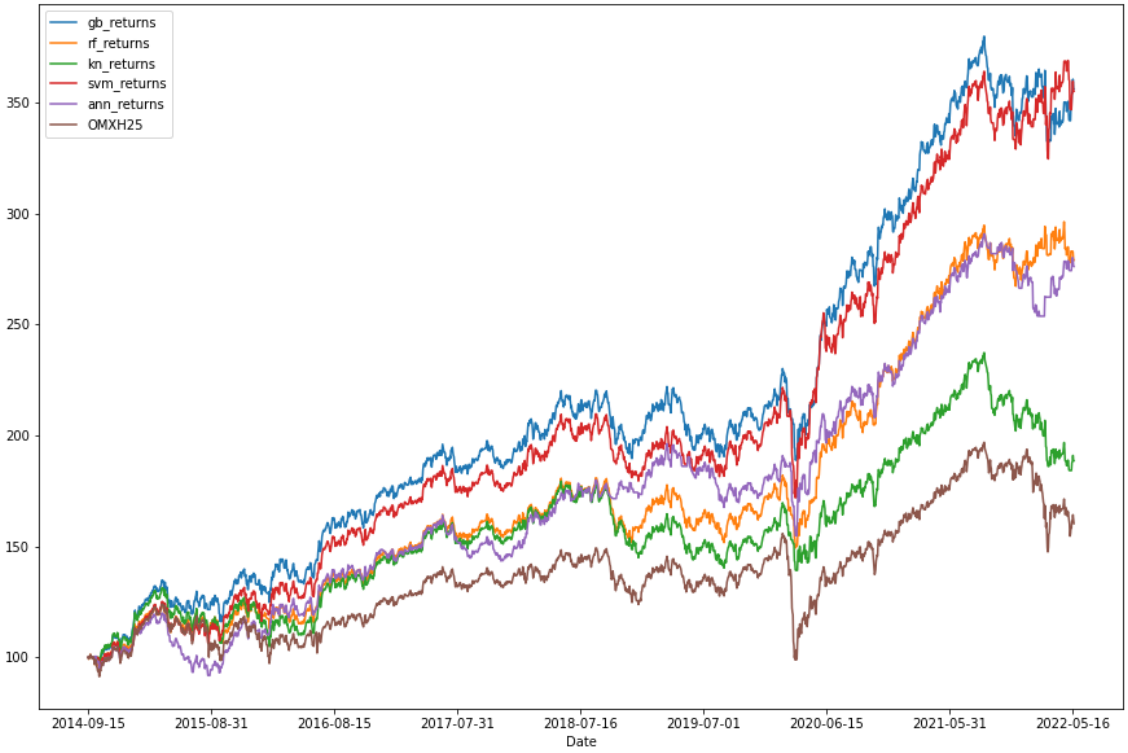

Figure 11. Model performance in trading simulation.

The results are similar to Nevasalmis (2020), where all the models beat the buy and hold benchmark. Likewise, in Nevasalmis study gradient boosting classifier performed the best of the considered machine learning methods.

7 Conclusions

This study builds on Nevasalmi (2020) and forecasts the OMXH25 daily returns using machine learning methods. Used machine learning methods include *k*-nearest neighbors, random forest, gradient boosting, support vector machine as well as feed-forward neural network. The collected dataset consists of 8014 daily observations between 1991 and 2022 including macroeconomic variables, data from other stock-exchanges and raw material markets as well as technical indicators derived from OMXH25 price.

The empirical results of this study show that machine learning methods are able to predict the daily returns of Finnish stock market. In statistical predictive performance the machine learning methods achieve 0,56 accuracies, excluding artificial neural network with accuracy of 0,54. Random forest classifier achieved the best accuracy of 0,5684. In economic predictive performance, measured with trading simulation, the models performances differ. Gradient boosting classifier and support vector machine achieve the best results, followed by random forest and artificial neural network. *k*-nearest neighbors performed the worst, but nevertheless all models were able to beat the benchmark buy and hold strategy. Top performer gradient boosting classifier gained returns of 355,98 percent during the trading simulation compared to 160,3 percent return of the benchmark.

These results indicate that efficient market hypothesis did not hold for the Finnish stock market and there were possibilities to gain excess returns using machine learning methods. With the continuous surge in computing power machine learning based methods could give investors opportunities for excess returns in the future.

References

Abdelmoula, A. K. (2015). Bank credit risk analysis with k-nearest-neighbor classifier: Case of Tunisian banks. *Accounting and Management Information Systems*, 14(1), 79.

Alkhatib, K., Najadat, H., Hmeidi, I., & Shatnawi, M. K. A. (2013). Stock price prediction using k-nearest neighbor (kNN) algorithm. *International Journal of Business, Humanities and Technology*, 3(3), 32-44.

Ampomah, E. K., Qin, Z., & Nyame, G. (2020). Evaluation of tree-based ensemble machine learning models in predicting stock price direction of movement. *Information,* 11(6), 332. https://doi.org/10.3390/info11060332

Ayala, J., García-Torres, M., Noguera, J. L. V., Gómez-Vela, F., & Divina, F. (2021). Technical analysis strategy optimization using a machine learning approach in stock market indices. *Knowledge-Based Systems*, 225. https://doi.org/10.1016/j.knosys.2021.107119

Ballings, M., Van den Poel, D., Hespeels, N., & Gryp, R. (2015). Evaluating multiple classifiers for stock price direction prediction. *Expert systems with Applications*, 42(20), 7046-7056. https://doi.org/10.1016/j.eswa.2015.05.013

Basak, S., Kar, S., Saha, S., Khaidem, L., & Dey, S. R. (2019). Predicting the direction of stock market prices using tree-based classifiers. *The North American Journal of Economics and Finance*, 47, 552-567. https://doi.org/10.1016/j.najef.2018.06.013

Baser, P., Saini, J. R., & Baser, N. (2023). Gold Commodity Price Prediction Using Tree-based Prediction Models. *International Journal of Intelligent Systems and Applications in Engineering,* 11(1), 90-96. https://www.ijisae.org/index.php/IJISAE/article/view/2481

Breiman, L. (2001). Random forests. *Machine learning*, 45, 5-32. https://doi.org/10.1023/A:1010933404324

Gaganis, C., Pasiouras, F., Spathis, C., & Zopounidis, C. (2007). A comparison of nearest neighbours, discriminant and logit models for auditing decisions. *Intelligent Sys-*

tems in Accounting, Finance & Management: International Journal, 15(1-2), 23-40.

Campisi, G., Muzzioli, S., & De Baets, B. (2023). A comparison of machine learning methods for predicting the direction of the us stock market on the basis of volatility indices. *International Journal of Forecasting* https://doi.org/10.1016/j.ijforecast.2023.07.002

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20, 273-297. https://doi.org/10.1007/BF00994018

Chollet, F. (2021). *Deep Learning with Python* (2). Manning.

Dias, R., Teixeira, N., Machova, V., Pardal, P., Horak, J., & Vochozka, M. (2020). Random walks and market efficiency tests: evidence on US, Chinese and European capital markets within the context of the global Covid-19 pandemic. *Oeconomia Copernicana*, 11(4), 585-608. https://doi.org/10.24136/oc.2020.024

Fama, E. F. (1965). The behavior of stock-market prices. *The journal of Business*, 38(1), 34-105.

Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The journal of Finance*, 25(2), 383-417.

Hamid, K., Suleman, M. T., Ali Shah, S. Z., & Imdad Akash, R. S. (2017). Testing the weak form of efficient market hypothesis: Empirical evidence from Asia-Pacific markets. https://dx.doi.org/10.2139/ssrn.2912908

Hastie, T., Tibshirani, R., & Friedman, J. (2017). *The Elements of statistical learning* (2). Springer.

Imandoust, S. B., & Bolandraftar, M. (2013). Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International journal of engineering research and applications*, 3(5), 605-610.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning* (2). Springer.

Khaidem, L., Saha, S., & Dey, S. R. (2016). Predicting the direction of stock market prices using random forest. *arXiv preprint*. https://doi.org/10.48550/arXiv.1605.00003

Khan, A. H., Shah, A., Ali, A., Shahid, R., Zahid, Z. U., Sharif, M. U., Tariqullah, J., & Zafar, M. H. (2023). A performance comparison of machine learning models for stock market prediction with novel investment strategy. *Plos one*, 18(9). https://doi.org/10.1371/journal.pone.0286362

Khan, W., Ghazanfar, M. A., Asam, M., Iqbal, A., Ahmad, S., & Khan, J. A. (2016). PREDICTING TREND IN STOCK MARKET EXCHANGE USING MACHINE LEARNING CLASSIFIERS. *Science International*, 28(2).

Lee, C. C., Lee, J. D., & Lee, C. C. (2010). Stock prices and the efficient market hypothesis: Evidence from a panel stationary test with structural breaks. *Japan and the world economy*, *22*(1), 49-58. https://doi.org/10.1016/j.japwor.2009.04.002

Liu, C., Chan, Y., Alam Kazmi, S. H., & Fu, H. (2015). Financial fraud detection model: Based on random forest. *International journal of economics and finance*, 7(7). https://doi.org/10.5539/ijef.v7n7p178

Lohrmann, C., Luukka, P. (2019). Classification of intraday S&P500 returns with a Random Forest. *International Journal of Forecasting*, 35(1), 390-407. https://doi.org/10.1016/j.ijforecast.2018.08.004

Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1), 59-82. https://doi.org/10.1257/089533003321164958

Metz, C. & Mickle, T. (2024, February 16). OpenAI Completes Deal That Values the Company at 80 Billion, *New York Times*. https://www.nytimes.com/2024/02/16/technology/openai-artificial-intelligence-deal-valuation.html

Mudassir, M., Bennbaia, S., Unal, D., & Hammoudeh, M. (2020). Time-series forecasting of Bitcoin prices using high-dimensional features: a machine learning approach. *Neural computing and applications*, 1-15. https://doi.org/10.1007/s00521-020-05129-6

Mukid, M. A., Widiharih, T., Rusgiyono, A., & Prahutama, A. (2018, May). Credit scoring analysis using weighted k nearest neighbor. In *Journal of Physics: Conference Series*, 1025(1). https://doi.org/10.1088/1742-6596/1025/1/012114

Nevasalmi, L. (2020) Essays on economic forecasting using machine learning. [dissertation, University of Turku] https://urn.fi/URN:ISBN:978-951-29-8223-3

Neely, C. J., Rapach, D. E., Tu, J., & Zhou, G. (2014). Forecasting the equity risk premium: the role of technical indicators. *Management science*, 60(7), 1772-1791.

Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques. *Expert Systems with Applications*, 42, 259–268. http://dx.doi.org/10.1016/j.eswa.2014.07.040.

Picasso, A., Merello, S., Ma, Y., Oneto, L., & Cambria, E. (2019). Technical analysis and sentiment embeddings for market trend prediction. *Expert Systems with Applications*, 135, 60-70. https://doi.org/10.1016/j.eswa.2019.06.014

Roy, S.S., Chopra, R., Lee, K.C., Spampinato, C. and Mohammadi-ivatlood, B. (2020). Random forest, gradient boosted machines and deep neural network for stock price forecasting: a comparative analysis on South Korean companies, *Int. J. Ad Hoc and Ubiquitous Computing*, 33(1), 62–71. https://doi.org/10.1504/IJAHUC.2020.104715

Sadorsky, P. (2021). A random forests approach to predicting clean energy stock prices. *Journal of Risk and Financial Management*, 14(2), 48. https://doi.org/10.3390/jrfm14020048

Sadorsky, P. (2021). Predicting gold and silver price direction using tree-based classifiers. *Journal of Risk and Financial Management*, 14(5), 198. https://doi.org/10.3390/jrfm14050198

Sadorsky, P. (2022). Forecasting solar stock prices using tree-based machine learning classification: How important are silver prices? *The North American Journal of Economics and Finance*, 61. https://doi.org/10.1016/j.najef.2022.101705

Scikit-learn, Retrieved May 7, 2024 from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

Shalf, J. (2020). The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166). https://doi.org/10.1098/rsta.2019.0061

Soepriyanto, B. (2021). Comparative Analysis of K-NN and Naïve Bayes Methods to Predict Stock Prices. *International Journal of Computer and Information System (IJCIS)*, 2(2), 49-53.

Subha, M. V., & Nambi, S. T. (2012). Classification of Stock Index movement using k-Nearest Neighbours (k-NN) algorithm. WSEAS Transactions on Information Science & Applications, 9(9), 261-270.

Teixeira, L. A., & De Oliveira, A. L. I. (2010). A method for automatic stock trading combining technical analysis and nearest neighbor classification. *Expert Systems with Applications*, 37(10), 6885–6890. https://doi.org/10.1016/j.eswa.2010.03.033

Toochaei, M. R., & Moeini, F. (2023). Evaluating the performance of ensemble classifiers in stock returns prediction using effective features. *Expert Systems with Applications*, 213. https://doi.org/10.1016/j.eswa.2022.119186

Valencia, F., Gómez-Espinosa, A., & Valdés-Aguirre, B. (2019). Price movement prediction of cryptocurrencies using sentiment analysis and machine learning. *Entropy*, 21(6), 589. https://doi.org/10.3390/e21060589

Weng, B., Lu, L., Wang, X., Megahed, F. M., & Martinez, W. (2018). Predicting short-term stock prices using ensemble methods and online data sources. *Expert Systems with Applications*, 112, 258-273. https://doi.org/10.1016/j.eswa.2018.06.016

Worasucheep, C. (2021). Ensemble Classifier for Stock Trading Recommendation. *Applied Artificial Intelligence*, 1–32. https://doi.org/10.1080/08839514.2021.2001178