Metódy strojového učenia aplikované pre investičné stratégie

Machine learning methods applied for investment strategies

Richard Martinus[1]

Abstrakt
Strojové učenie sa čoraz častejšie stáva kľúčovým a preferovaným nástrojom pri tvorení a optimalizácii investičných portfólií. Jeho využitie spočíva v presnejších predikciách, adaptívnom rozhodovaní a vlastnosti naučiť sa podľa vzorov ako reagovať na trhové udalosti. V oblasti financií je strojové učenie uplatňované pomocou regresných modelov, neurónových sietí alebo aj učením formou odmeňovania, kde cieľom je predpovedať cenové pohyby aktív. Využitím neurónových sietí je možná identifikácia komplexných vzorov v historických dátach, zatiaľ čo učenie na základe odmeňovania optimalizuje stratégie obchodovania podľa spätnej väzby. V článku poukazujeme aj na výhody a výzvy spojené s používaním strojového učenia v investovaní ako je interpretovateľnosť modelov alebo kvalita dát.

Kľúčové slová
Strojové učenie, portfólio, investičné stratégie

Abstract
Machine learning is increasingly becoming a key and preferred tool in the creation and optimization of investment portfolios. Its use lies in more accurate predictions, adaptive decision-making and the ability to learn from patterns how to respond to market events. In the field of finance, machine learning is applied using regression models, neural networks or even reinforcement learning, where the goal is to predict asset price movements. Using neural networks, it is possible to identify complex patterns in historical data, while reinforcement learning optimizes trading strategies based on feedback. We also discuss the advantages and challenges associated with the use of machine learning in investing, such as model interpretability or data quality.

Key words
Machine learning, portfolio, investing strategies

JEL classification
C45, C53, G11, G17

[1] Bratislava University of Economics and Business, Faculty of Economic Informatics, Department of Operations Research and Econometrics, Dolnozemská cesta 1, 852 35 Bratislava, richard.misek@euba.sk.

1 Introduction

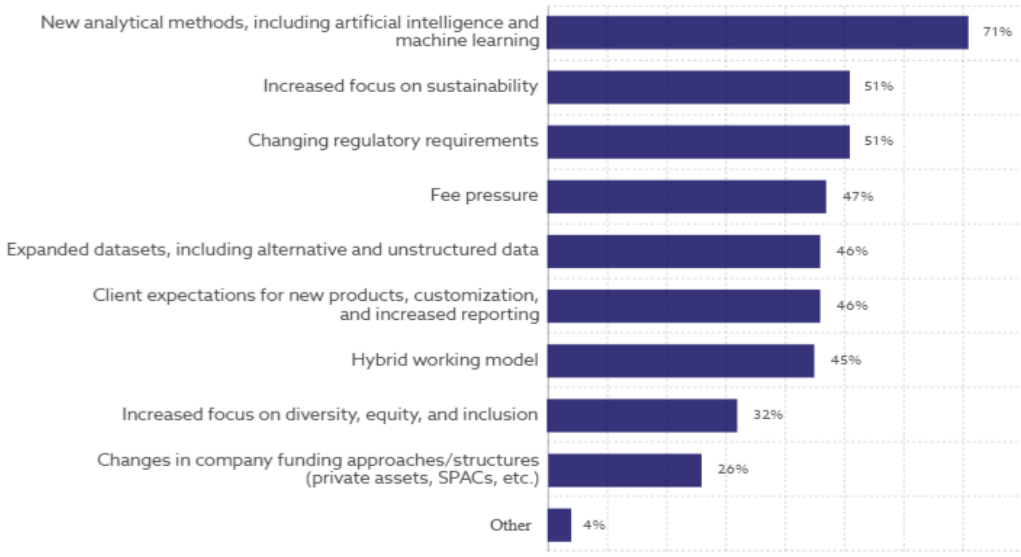
Learning is the foundation of every human being from birth, starting with learning to eat, taking the first steps, and continuing through school. Learning can be used not only in humans, but also in technology. In the world of information technology, machine learning (ML) can be defined as the science of computers' ability to learn without being explicitly programmed. Machine learning is a subset of artificial intelligence (AI) and is an impressive way to approach solving complex problems. This revolutionary approach to analyzing, processing, and interpreting data allows computer systems to learn and subsequently improve in the area they are used for. Improvement in machine learning is the ability to improve the results of an algorithm with minimal human intervention. Unlike traditional programming, where a programmer manually defines rules, machine learning allows a system to adapt to new situations based on existing data. There are many algorithms and different machine learning techniques, the most common of which are supervised learning, unsupervised learning, and reward-based learning. In supervised learning, a model is trained on labeled data, where the outputs are labeled. The goal here is to teach the model to predict outputs based on new inputs. Unsupervised learning, as the name suggests, is based on learning from unlabeled input data. The goal is to discover patterns and groups or structures in the data that are related to each other. Reinforcement learning is based on learning from interaction with the environment. The goal is to maximize reward or minimize loss, and this type can be imagined by awarding points for good decisions and deducting points for bad decisions. Various techniques can also be used in predicting financial markets, which is generally a difficult, if not impossible, discipline. It is never possible to accurately predict events such as inflation, natural disasters, pandemics or other macroeconomic events that can change the market trend. Analyses can only approximate the future market direction and the analyses created should only form guidelines for predicting asset prices. We chose the topic of machine learning because it is an increasingly attractive approach to solving complex problems not only in finance and allows for continuous improvement of the algorithm. This article is focused on a deeper understanding of machine learning, its methods and techniques in the field of finance, where we will illustrate the application of machine learning in predicting the prices of financial assets. The goal is to examine how machine learning can determine the future value of an asset or at least approach the value that may occur in the future.

2 Literature review

Machine learning and artificial intelligence are rapidly becoming mainstream approaches to solving various investment strategies. Its history dates back to the 1950s, where the most famous article that is considered the foundation of machine learning is “Computing Machinery and Intelligence” (Turing, 1950). Turing was concerned with the question of whether machines can think and introduced the Turing test, which evaluates a machine’s intelligence according to its ability to communicate like a human. Here he outlined the idea that machines could learn from data and experience, just like humans. The first program that learned from experience was created for the game of checkers, where learning was used in the form of rewards and heuristic search. The program improved here, so that it played against itself (Samuel, 1959). Subsequently, new approaches and contributions were developed and discovered that addressed the topic of machine learning. When it comes to investing, there are several approaches that can be used, such as active trading, where the investor buys and sells assets over a short period of time (days or weeks), but it can also be a long-term investment strategy. In this chapter, we will present several articles studied abroad, in which we will describe the solution procedure that the authors chose to solve this problem. One of the first articles focused on investing using machine learning is a article from the 1980s that applied neural networks to price prediction

and was constructed when predicting IBM stocks. The author focused on the solution using MLP multilayer perceptrons, which he used to predict stock prices (White, 1988). Among the first articles is the valuation and hedging of derivative securities using machine learning methods. The article focused on non-parametric methods such as neural networks. The authors' approach enabled a more accurate approach to modeling nonlinear patterns in markets, while a new tool was created for optimizing hedging strategies (Lo & Poggio, 1994). The article "Application of machine learning in algorithmic investment strategies on global stock markets" focuses on the use of machine learning in algorithmic investment strategies. Several models are analyzed here, such as neural networks, naive Bayesian models or regression trees for generating trading signals on the Warsaw WIG20 index, the German DAX index and the American S&P500 index and selected Central and Eastern European indices in the period from January 2002 to the end of March 2023. Individual strategies and the benchmark buy-and-hold strategy were compared with each other, where risk and return were compared. The results showed that machine learning models outperformed passive strategies, with the best performing models being the polynomial support vector machine for the WIG20 and S&P500 indices. The best performing model for the DAX index was the linear support vector machine (Grudniewicz & Ślepaczuk, 2023). Another study presents financial market indicators that can be used for investment strategies when investing in the global stock market. The authors propose to create a design of unmanaged and managed global stock market volatility using pairwise correlation and vector autoregressive models. The effectiveness of these indicators was analyzed using logistic regression, SVM, and the random forest algorithm. Strategies for global stock market prediction and regional allocation for emerging markets were proposed. The results showed that these indicators are useful in times of crisis. The study is one of the first attempts to apply financial indicators in investment strategies (Lee & Cho & Kwon & Sohn, 2018). Focusing on the development of a sustainable quantitative investment model is also possible using machine learning. The model that the authors investigated includes stock selection and algorithmic trading, using principal component analysis and EVA (Economic Value Added) indicators for stock selection. Testing the model on the US stock market showed that the LSTM networks can predict stock prices more accurately from the used techniques. The strategy generated higher returns than the market in all market conditions. (Li & Wang & Ahmad & Huang & Khan, 2023). We will also practically apply the LSTM technique in this article. Studies are often focused on long-term investment strategies within Value Investing. This approach is based on estimating the intrinsic value of an individual company and investing in undervalued stocks, which represents a safety margin. The goal of this study was to automate stock selection by combining the principles of Value Investing and historical financial data. The classification models used classify stocks based on investment activity. The model's success rate reached more than 80%, identifying stocks with a potential for 15% annualized returns over a three-year horizon from the recommended purchase date. (Priel & Rokach, 2024). The overall benefit and increasing use of machine learning is also illustrated in a survey by the CFA Institute conducted in 2022, where 71% of respondents stated that new analytical methods created by artificial intelligence and machine learning will significantly affect job positions in the next 5 to 10 years. An overall overview can be seen in Figure 1.

Figure 1: CFA survey



Source: CFA Institute

3 Methodology and research methods

Machine learning is based on the idea that a computer should be able to learn independently without human assistance and then be able to apply what it has learned in practice. This chapter will introduce three categories of learning methods (supervised learning, unsupervised learning, and reinforcement learning), and will also introduce the algorithms that are often used in these types, the principle of testing and training data, which form the basic building blocks of machine learning, and measures of forecast accuracy.

3.1 Data selection

Testing and training data are the basis for machine learning, in achieving the correctness of algorithms and can be divided into two sets. **Training data set** represents the first series of data that is used in the system. This data is used to improve or train the model and usually represents between 70% and 80% of the total data set. It is therefore used to train machine learning algorithms so that they can make predictions, which can be for example classifications based on relationships and patterns found in this data set. When improving the model, we recognize inputs that represent characteristics or attributes that the model analyzes. An example could be predicting house prices, where the inputs can be location, plot size or number of rooms. The inputs are followed by outputs that represent target values or categories that we want the model to predict. If we were to stick to the example above, then for house prices the output would be the price of the house. The quality and size of the training data set strongly affects the performance of a machine learning model. In general, the training process consists of data collection and preparation, data partitioning, model training, model evaluation, and model optimization. Without training data, models would not be able to learn and generate predictions of results such as classification. **Testing data set** is the second data set, which usually represents 20% to 30% of the data from the total set. This data is just as important as the training data, although its use in the time sequence is after the training data. It is used to evaluate the performance of the model after it has been trained. Evaluation consists of verifying whether the model can correctly predict or classify new data and thus assess its ability to generalize. Test data also consists of inputs and outputs. Inputs represent characteristics or attributes that the model analyzes. These inputs can be, for example, measurements, parameters, or other relevant information. Outputs are target values or categories that the model predicts. They are used to compare with model predictions to evaluate the accuracy of the model. When testing data, the

usual first step is to divide the data into a set, evaluate the model, followed by evaluation methods and generalization with validation. We distinguish several evaluation methods, for example, accuracy, precision, recall, F1-score, measures of forecast accuracy such as mean square error (MSE), root mean square error (RMSE) or mean absolute error (MAE). We will discuss quality assessment at the end of this chapter. Testing data allows us to assess whether a trained model is overfitted or underfitted and is crucial for objective model evaluation (Chatzilygeroudis & Hatzilygeroudis & Perikos, 2021). Important issues in model evaluation, optimization, and selection are the overfitting and underfitting. Each of these issues represents an extreme in the model's ability to learn from training data and generalize correctly to new, unknown data. **Undertraining** occurs when a model is unable to capture patterns and relationships in the training data well. The model is too simple, and its prediction capabilities are weak. This leads to low accuracy on both training and test data. The first cause may be that the model is too simple, whereas if the model is not complex enough, it cannot capture patterns in the data. Other causes include a lack of features (input variables), too much regularization, which penalizes large weights and leads to a model that is too simple, or a lack of training data. An undertrained model is unable to properly capture the essence of the data and has low learning from the training data. Undertraining can be avoided by using a more complex model that better captures patterns in the data, adding relevant features, reducing the degree of regularization, or increasing the amount and quality of training data. **Overtraining** occurs when a model learns training data too well, even with the noise of this data and anomalies, which leads to high accuracy on training data, but poor generalization to new unknown data. Common causes of overtraining are an overly complex model with many parameters that adapt too well to the training data. To avoid overtraining, measures can be taken such as using regularization techniques that penalize large weights such as L1 and L2, using cross-validation to evaluate the model, reducing the complexity of the model, e.g. by reducing the number of parameters or layers in the neural netarticle, increasing the amount and quality of training data (Demyanov, 2015).

3.2 Types of learning in ML

Supervised learning is one of the main approaches in machine learning, where models learn from data that contains input and output pairs. The model is trained on training data that contains inputs and their corresponding outputs. The goal of the model is to learn the mapping between inputs and outputs so that it can make predictions about unknown and future data (target or explanatory variable). A prerequisite for its use is the existence of a sufficiently large training dataset that contains correctly labeled input–output pairs. It is therefore strongly dependent on a sufficient amount of quality data. In supervised learning, we distinguish two types of supervised learning: classification and regression. The difference in **unsupervised learning** is that the data is not labeled, and the algorithms try to discover, model, and describe patterns in the data in order to learn new information. The most used algorithm is clustering. In the case of **reinforcement learning**, no labeled or unlabeled training examples are used to train the model. Reinforcement learning occurs by creating a system (agent), which we deploy into the environment and let it learn through interaction with the environment. The only thing we need to determine about it are the rules for how it can behave in a given environment and the so-called reward function. With its help, the agent can evaluate whether the decision it has just made was beneficial for it or not. Then, using the trial-and-error method, like a human, it tries out individual options and learns how to ideally behave in individual situations. A model example for reinforcement learning is the game of chess - Deep blue defeated Kasparov, 1997. Detailed information can be found in the article Reinforcement Learning: an Overview (Glorennec, 2000).

3.3 Forecast accuracy rates

The accuracy measures are based on a measure of the forecast error – the difference between the actual and estimated values $e_i = y_i - \hat{y}_i$ (where e_i is called the residual). The goal is to minimize forecast errors, that is, the better model is the one with the lower value of the calculated forecast accuracy measure. We do this with the following two formulas:

The **mean square error** (MSE) is used to measure the average size of the errors in the forecasts, where more emphasis is placed on larger errors. Lower MSE values predict better model quality. MSE is sensitive to extreme values, since larger errors have a quadratic effect on the result.

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

The **root mean square error** (RMSE) is a metric derived from the MSE. Lower RMSE values (as with MSE) indicate a better model. RMSE places greater weight on large errors. RMSE is used to determine how far predictions are, on average, from the true values.

$$RMSE = \sqrt{MSE} \quad (2)$$

4 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a type of recurrent neural netarticle (RNN) specifically designed to solve problems in sequential data processing, where the outputs of previous steps are used as inputs for subsequent steps. This approach allows RNNs to retain information about previous states and process data with time dependencies. RNNs are useful for tasks such as time series, natural language processing, and speech recognition. LSTM is used in similar areas such as natural language processing, speech recognition, and time series. In natural language processing, LSTMs are used for tasks such as text translation, text generation, and sentiment analysis. In speech recognition, LSTMs are used to convert speech to text. Time series prediction is the prediction of future values based on historical data, and for example, music generation is based on learning musical sequences and creating new compositions. With LSTM, the goal is to provide a neural netarticle with a short-term memory spanning thousands of gradient steps, which represents a long short-term memory. This netarticle can be used in tasks that use memories of events that happened many (thousands and millions) steps earlier. LSTM solves problems with long-term dependencies that recurrent neural networks cannot handle due to the problem of vanishing gradients. LSTM allows for efficient identification and retention of relevant information over long periods of time, which is crucial for successful processing of sequential data. With LSTM, we recognize basic properties such as cells and gates (Sherstinsky, 2020).

A **cell** remembers values at any given time interval, and three gates regulate the flow of information into and out of the cell. An LSTM is made up of cells, which are the basic building blocks of the netarticle. A cell has an internal state that can be transmitted over many time steps, allowing the LSTM to "remember" information over long sequences.

Three main gates are used in an LSTM to control information:

The **input gate** decides which pieces of new information are stored in the current state, using the same system as the forget gates, in other words, it determines which new information is added to the cell.

The **forget gate** decides which information from the previous state to discard, by assigning a value between 0 and 1 to the previous state. A value of 1 means to keep the information, and a value of 0 means to discard the information. It therefore decides which information from the previous state is to be forgotten.

The **output gate** controls which information in the current state is to be output by assigning a value from 0 to 1 to the information with respect to the previous and current state. It thus determines which part of the cell state will be used as output.

The most common type of learning for LSTM is supervised learning, where the model is trained on labeled data. The application here is for time series prediction, where predicting future values is based on historical data, for example, predicting stock prices. Natural language processing is represented by tasks such as machine translation, text generation, sentiment analysis, or speech-to-text recognition. Sequence classification can be, for example, sequence classification (spam filter) or audio signal categorization.

5 Machine learning application with LSTM approach

Predicting the exact development of securities prices is an impossible discipline. The factors that affect a company's price on the market are mainly economic indicators, and only then can price predictions be considered. In this example, the price is determined without considering macroeconomic factors - without adapting fundamental analysis. The result is a price determination for the next day, based on the data provided. This data is drawn from Yahoo Finance. We articleed with a ten-year period from May 15, 2014, to May 15, 2024, and the analyzed company was the French company LVMH Moët Hennessy - Louis Vuitton S.A. The first step was to import libraries and create an empty data frame, into which we downloaded the price for the given period using the Yahoo Finance API (application programming interface) and found out what shape or size our data is. The data matrix represents 2561 rows and 6 columns and is shown in Figure 2.

Figure 2: Filling data into a data frame

```
# Import necessary libraries for data manipulation and visualization
import math
import pandas_datareader as web  # For accessing financial data
import numpy as np  # For numerical computations
import pandas as pd  # For data manipulation
from sklearn.preprocessing import MinMaxScaler  # For normalizing data
from keras.models import Sequential  # For building the neural network
from keras.layers import Dense, LSTM  # For adding layers to the neural network
import matplotlib.pyplot as plt  # For plotting graphs
plt.style.use('fivethirtyeight')  # For setting the plot style
import yfinance as yf  # For accessing financial data from Yahoo Finance
```

```
# Create an empty DataFrame to store the data
df = pd.DataFrame()

# Fetch historical stock price data for the ticker 'ASME.DE' from Yahoo Finance
df = yf.download('MC.PA', start='2014-05-15', end='2024-05-15')

# Display the fetched data
df
```

```
[*********************100%***********************]  1 of 1 completed
```

Date	Open	High	Low	Close	Adj Close	Volume
2014-05-15 00:00:00+02:00	142.250000	144.649994	141.899994	143.550003	120.781311	1157770
2014-05-16 00:00:00+02:00	143.850006	144.149994	142.550003	143.750000	120.949539	812583
2014-05-19 00:00:00+02:00	143.750000	143.949997	141.600006	142.350006	119.771629	797627
2014-05-20 00:00:00+02:00	142.699997	143.000000	141.250000	142.750000	120.108192	528324
2014-05-21 00:00:00+02:00	142.600006	143.949997	141.699997	143.949997	121.117836	807472
...	...	...	...	...	...	...
2024-05-08 00:00:00+02:00	790.000000	795.700012	786.000000	787.900024	787.900024	204110
2024-05-09 00:00:00+02:00	785.000000	789.000000	778.099976	789.000000	789.000000	150180
2024-05-10 00:00:00+02:00	795.099976	796.400024	787.000000	788.400024	788.400024	303924
2024-05-13 00:00:00+02:00	792.599976	793.099976	782.099976	782.599976	782.599976	167046
2024-05-14 00:00:00+02:00	783.599976	791.900024	780.099976	791.900024	791.900024	196008

2561 rows × 6 columns

```
# Get the number of rows and columns in the dataset
rows, columns = df.shape

# Print the number of rows and columns
print(f"The dataset contains {rows} rows and {columns} columns.")
```

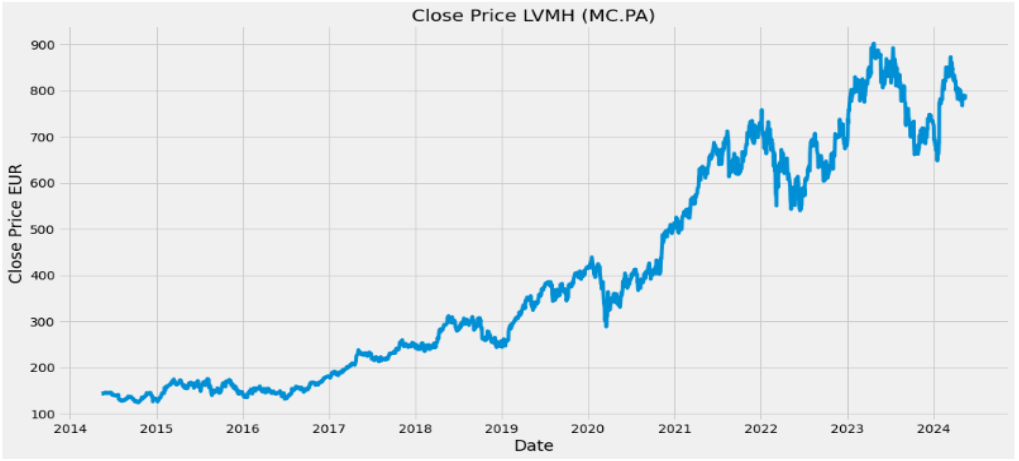
```
The dataset contains 2561 rows and 6 columns.
```

Source: own processing

We then filtered out the closing price and displayed it on the chart using the plt.show() function. We labeled both axes and named the chart, which we show in Figure 3.

Figure 3: Plotting 10 years closing price of LVMH

```
# Set the figure size for the plot
plt.figure(figsize=(16,8))

# Set the title of the plot
plt.title('Close Price LVMH (MC.PA)')

# Plot the closing price history
plt.plot(df['Close'])

# Set the label for the x-axis
plt.xlabel('Date', fontsize=18)

# Set the label for the y-axis
plt.ylabel('Close Price EUR', fontsize=18)

# Display the plot
plt.show()
```



Source: own processing

Then, we create a new dataframe in Figure 4, which will contain only the closing price, and convert the dataframe to an array. After the conversion, we set the training data range to 80% and print out their number (2049). Using the MinMaxScaler function, we scale the data so that all values lie within the specified range, and then we train the scaling model on the dataset, which is where the data transformation occurs.

Figure 4: Creating training data sets

```
# Create a new DataFrame with only the 'Close' column
data = df.filter(['Close'])

# Convert the DataFrame to a numpy array
dataset = data.values

# Get the number of rows to train the model on (80% of the dataset)
training_data_len = math.ceil(len(dataset) * 0.8)

# Print the number of training rows
training_data_len
```

```
2049
```

```
# Initialize the MinMaxScaler with the feature range (0, 1)
scaler = MinMaxScaler(feature_range=(0, 1))

# Fit the scaler to the dataset and transform the data
scaled_data = scaler.fit_transform(dataset)

# Display the scaled data
scaled_data
```

```
array([[0.0254417 ],
       [0.02569868],
       [0.02389978],
       ...,
       [0.85403151],
       [0.84657883],
       [0.85852878]])
```

Source: own processing

After the MinMaxScaler function, we create a training dataset, assigning the previous 60 days to the x_train dataset and assigning the current day to the y_train variable.

Figure 5: Creating a training dataset and converting it to numpy arrays

```
# Create the training dataset
# Create the scaled training data set
train_data = scaled_data[0:training_data_len, :]

# Initialize the training data sets
x_train = []
y_train = []

# Split the data into x_train and y_train datasets
for i in range(60, len(train_data)):
    # Append the past 60 days of data to x_train
    x_train.append(train_data[i-60:i, 0])
    # Append the current day (i-th day) to y_train
    y_train.append(train_data[i, 0])
    # Print the first few sets for verification
    if i <= 61:
        print("x_train:")
        print(x_train)
        print("y_train:")
        print(y_train)
        print()

# Convert x_train and y_train to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)
```

Source: own processing

At the figure 6 we show output of the created arrays.

Figure 6: Creating a training dataset and converting it to numpy arrays

```
x_train:
[array([0.0254417 , 0.02569868, 0.02389978, 0.02441375, 0.02595567,
       0.0254417 , 0.02492771, 0.02634115, 0.02685512, 0.0262769 ,
       0.02666238, 0.02852553, 0.02691937, 0.0273691 , 0.02762608,
       0.02826855, 0.02505622, 0.0253132 , 0.02679088, 0.02717635,
       0.02698362, 0.02653388, 0.02646965, 0.02781882, 0.02711212,
       0.0282043 , 0.0282043 , 0.02794732, 0.02839705, 0.02563443,
       0.02248635, 0.02132991, 0.02190813, 0.02216511, 0.02081593,
       0.02325732, 0.02197238, 0.02043045, 0.01888853, 0.02023771,
       0.02004498, 0.02004498, 0.02132991, 0.01920976, 0.02088018,
       0.01901703, 0.01882428, 0.01888853, 0.01991648, 0.0206232 ,
       0.02248635, 0.01015097, 0.01002249, 0.01021522, 0.00835207,
       0.00623194, 0.0057822 , 0.00475425, 0.0077096 , 0.00732412])]
y_train:
[0.004111785349492053]

x_train:
[array([0.0254417 , 0.02569868, 0.02389978, 0.02441375, 0.02595567,
       0.0254417 , 0.02492771, 0.02634115, 0.02685512, 0.0262769 ,
       0.02666238, 0.02852553, 0.02691937, 0.0273691 , 0.02762608,
       0.02826855, 0.02505622, 0.0253132 , 0.02679088, 0.02717635,
       0.02698362, 0.02653388, 0.02646965, 0.02781882, 0.02711212,
       0.0282043 , 0.0282043 , 0.02794732, 0.02839705, 0.02563443,
       0.02248635, 0.02132991, 0.02190813, 0.02216511, 0.02081593,
       0.02325732, 0.02197238, 0.02043045, 0.01888853, 0.02023771,
       0.02004498, 0.02004498, 0.02132991, 0.01920976, 0.02088018,
       0.01901703, 0.01882428, 0.01888853, 0.01991648, 0.0206232 ,
       0.02248635, 0.01015097, 0.01002249, 0.01021522, 0.00835207,
       0.00623194, 0.0057822 , 0.00475425, 0.0077096 , 0.00732412]), array([0.02569868, 0.02389978, 0.02441375, 0.02595567, 0.0
254417 ,
       0.02492771, 0.02634115, 0.02685512, 0.0262769 , 0.02666238,
       0.02852553, 0.02691937, 0.0273691 , 0.02762608, 0.02826855,
       0.02505622, 0.0253132 , 0.02679088, 0.02717635, 0.02698362,
       0.02653388, 0.02646965, 0.02781882, 0.02711212, 0.0282043 ,
       0.0282043 , 0.02794732, 0.02839705, 0.02563443, 0.02248635,
       0.02132991, 0.02190813, 0.02216511, 0.02081593, 0.02325732,
       0.02197238, 0.02043045, 0.01888853, 0.02023771, 0.02004498,
       0.02004498, 0.02132991, 0.01920976, 0.02088018, 0.01901703,
       0.01882428, 0.01888853, 0.01991648, 0.0206232 , 0.02248635,
       0.01015097, 0.01002249, 0.01021522, 0.00835207, 0.00623194,
       0.0057822 , 0.00475425, 0.0077096 , 0.00732412, 0.00411179])]
y_train:
[0.004111785349492053, 0.0037263109873162947]
```

Source: own processing

We then convert the training data to numpy arrays and display the shapes of the training sets. We need to create a three-dimensional shape in order to use the LSTM model, the creation of which is shown in Figure 7.

Figure 7: Use of LSTM

```
# Convert the x_train and y_train lists to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Display the shapes of the resulting numpy arrays
print(f"x_train shape: {x_train.shape}")
print(f"y_train shape: {y_train.shape}")

# Reshape the x_train data to be 3-dimensional for the LSTM model
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

# Display the shape of the reshaped x_train data
x_train.shape
```

```
x_train shape: (1989, 60)
y_train shape: (1989,)

(1989, 60, 1)
```

```
Initialize a Sequential model
del = Sequential()

Add the first LSTM layer with 50 units, return_sequences=True since we are adding more LSTM layers after this one, and input_
del.add(LSTM(50, return_sequences=True, input_shape=(x_train.shape[1], 1)))

Add the second LSTM layer with 50 units and return_sequences=False since this is the last LSTM layer
del.add(LSTM(50, return_sequences=False))

Add a Dense layer with 25 units
del.add(Dense(25))

Add a Dense layer with 1 unit (output layer)
del.add(Dense(1))
```

```
#Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
#Train the model
model.fit(x_train, y_train, batch_size=1, epochs=1)
```

```
1989/1989 [==============================] - 37s 18ms/step - loss: 2.3962e-04

<keras.callbacks.History at 0x1f8d82a64f0>
```

```
# Select the scaled data from training_data_len - 60 to the end to create the testing dataset
test_data = scaled_data[training_data_len - 60:, :]

# Create empty lists to store the features (x_test) and target values (y_test) for testing
x_test = []
y_test = dataset[training_data_len:, :]  # Target values for testing data (unscaled)

# Populate x_test with sequences of 60 days for testing
for i in range(60, len(test_data)):
    # Append sequences of past 60 days for each testing data point to x_test
    x_test.append(test_data[i - 60:i, 0])

```

```
#Convert the data into a numpy array
x_test = np.array(x_test)
```

```
#Reshape the data to 3D shape
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

Source: own processing

We run the model prediction and calculate the RMSE (2), which currently gives a value of 3.9358. In Figure 7, we see the data split into training and validation and plot the output along with the real data. We see that the prediction was successful, and the price was set with a high probability at the real value.

Figure 8: Visualization of model outputs

```
# Get the model's predicted price values
predictions = model.predict(x_test)

# Inverse transform the scaled predictions to the original scale
predictions = scaler.inverse_transform(predictions)
```

```
16/16 [==============================] - 0s 12ms/step
```

```
#Get the root mean squared error (RMSE)
rmse = np.sqrt( np.mean(predictions - y_test)**2 )
rmse
```

```
3.935848593711853
```

```
# Split the data into train and valid sets
train = data[:training_data_len]
valid = data[training_data_len:].copy()  # .copy() to avoid SettingWithCopyWarning

# Add the predictions to the valid set
valid['Predictions'] = predictions

# Visualise the data
plt.figure(figsize=(16,8))
plt.title('Model')
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close Price USD', fontsize=18)
plt.plot(train['Close'])  # Plotting the training data
plt.plot(valid[['Close', 'Predictions']])  # Plotting the actual and predicted values for the validation data
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')  # Adding legend
plt.show()
```

Source: own processing

The results shown in Figure 9 show the price prediction versus the actual closing prices of the asset.

Figure 9: Example of predicted values versus actual closing prices

```
#Show the valid and predicted prices
valid
```

Date	Close	Predictions
2022-05-16 00:00:00+02:00	575.000000	595.800293
2022-05-17 00:00:00+02:00	590.099976	588.232117
2022-05-18 00:00:00+02:00	578.500000	603.457947
2022-05-19 00:00:00+02:00	568.500000	590.343201
2022-05-20 00:00:00+02:00	556.599976	579.988098
...	...	...
2024-05-08 00:00:00+02:00	787.900024	794.584839
2024-05-09 00:00:00+02:00	789.000000	791.004456
2024-05-10 00:00:00+02:00	788.400024	792.454651
2024-05-13 00:00:00+02:00	782.599976	791.254211
2024-05-14 00:00:00+02:00	791.900024	784.652222

512 rows × 2 columns

Source: own processing

The final step is to run the algorithm to predict the next value that the model has not yet been trained on internally. The results differ but remain close to real value with a partial deviation. The procedure is shown in Figure 10.

Figure 10: Running the prediction algorithm

```
# Fill in company details into a table
lvmh_quote = pd.DataFrame()

# Download historical stock price data for the 'MC.PA' ticker (LVMH) from Yahoo Finance
lvmh_quote = yf.download('MC.PA', start='2014-05-15', end='2024-05-15')

# Filter the DataFrame to include only the 'Close' column
lvmh_quote = lvmh_quote.filter(['Close'])

# Print the data for verification
print(lvmh_quote)

# Get the last 60 days of closing prices and scale the data
last_60_days = lvmh_quote[-60:].values
last_60_days_scaled = scaler.transform(last_60_days)

# Create an empty list and append the past 60 days
X_test = []
X_test.append(last_60_days_scaled)

# Convert X_test to a numpy array and reshape the data
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Get the predicted scaled price
pred_price = model.predict(X_test)

# Undo the scaling to get the predicted price in the original scale
pred_price = scaler.inverse_transform(pred_price)

# Print the predicted price for next day
print(pred_price)
```

```
[**********100%**********]  1 of 1 completed
                                 Close
Date
2014-05-15 00:00:00+02:00  143.550003
2014-05-16 00:00:00+02:00  143.750000
2014-05-19 00:00:00+02:00  142.350006
2014-05-20 00:00:00+02:00  142.750000
2014-05-21 00:00:00+02:00  143.949997
...                               ...
2024-05-08 00:00:00+02:00  787.900024
2024-05-09 00:00:00+02:00  789.000000
2024-05-10 00:00:00+02:00  788.400024
2024-05-13 00:00:00+02:00  782.599976
2024-05-14 00:00:00+02:00  791.900024

[2561 rows x 1 columns]
1/1 [==============================] - 0s 31ms/step
[[796.30865]]
```

```
# Fill in company details into a table
lvmh_quote2 = pd.DataFrame()

# Download data for 'ASML' ticker for a specific date
lvmh_quote2 = yf.download('MC.PA', start='2024-05-15', end='2024-05-16')

# Check if data was retrieved successfully
if not lvmh_quote2.empty:
    # Filter the DataFrame to include only the 'Close' column
    lvmh_quote2 = lvmh_quote2.filter(['Close'])

    # Display the 'Close' price for the specified date
    print(lvmh_quote2['Close'])
else:
    print("No data found for the specified date range. The symbol may be delisted or unavailable.")
```

```
[**********100%**********]  1 of 1 completed
Date
2024-05-15 00:00:00+02:00    789.0
Name: Close, dtype: float64
```

Source: own processing

This completes the code that predicts the future value of an asset based on data from the past ten years. The algorithm can be further improved using regularization techniques and validation, which may be considered in the future when optimizing this example.

Conclusion

At the end of the article, we would like to evaluate the results, the main findings and the progress of the article. At the beginning of the article, we set ourselves the goal of examining how it is possible to determine the future value of an asset using machine learning or at least get closer to the value that may occur in the future. In order to be able to work on these findings, we gradually introduced the types of learning we recognize in the methodology. We introduced the basic algorithms for these types of learning and determined the occasions when they are appropriate to be used. Subsequently, we introduced LSTM, which is key in the calculations performed by our algorithms. After defining and explaining these terms, we proceeded to examine the prediction of stock price development using machine learning. Machine learning uses testing, training data and LSTM. The work was carried out with the company LVMH Moët Hennessy. In this article, we focused on the European market, since the company is French. In the model, we used a ten-year history, where the indicative data were the closing daily values of LVMH. Based on these values, we created a model using an algorithm that tries to predict the future value of the asset and plots the predicted values against the actual values from the past. The created model consists of machine learning and is focused on predicting stock price. Asset prices can never be fully predicted and thus serve only as an aid in predicting price development. The model does not include other elements of either technical or fundamental analysis. The principle on which the model articles is LSTM. The results show that the predicted price is close to real prices and the resulting code presents that the predicted price is also close to real values. An improvement of this model in the future could be the application of multiple epochs, the application of regularization techniques or a change in the algorithm, which could then be compared with the results of other models. Improvement could also lie in the application of quarterly or annual reports of the company. If these reports are positive, they can greatly influence the price upwards, and in the case of negative results on economic growth and the company's intended expansion, the price can drop sharply by many percent. These aspects are not included in the model. Another possible improvement is the inclusion of a reaction to resistance and support zones, which are mostly reflection zones if they are reached (the price reflects from them in the reverse direction). The last significant improvement could also be the reaction to lowering or increasing interest rates together with inflation reports, which are superior to the technical analysis of the company. When incorporating and creating such a complicated model, it would be possible to get closer to the real price of assets, but it is still true that the model cannot be created in such a way that it accurately creates future asset prices. In conclusion, we can say that machine learning is becoming an integral part of our everyday life and its importance is constantly growing. Uses and applications can be found from healthcare to finance. Machine learning is revolutionizing the way we approach problems and make decisions. This method is still under development and the longer it is studied, the more accurate results we will be able to achieve using various algorithms. From the perspective of the future of machine learning, it is clear that development will not stop. Progress in this area can be seen especially in recent years from companies such as Nvidia, Alphabet, Meta or Tesla. These companies are gradually becoming major players in the world of machine learning and are thus able to use their resources and improve processes better. Machine learning has the potential to improve the quality of our daily lives, but it can also solve global problems. The work is fully applicable to other assets as well, not just one selected stock. The same principle is used for other assets or funds that are composed of multiple assets. In cases where multiple assets are analyzed in parallel, the program can be modified to add various securities. If we later attempt to model a portfolio from these stocks, it is necessary to combine them into a portfolio before the calculation, the price of which will then be predicted.

We believe that with further development and innovation, machine learning will continue to advance and will play a key role in shaping a better and more sustainable world.

Funding: This article was supported by the Grant Agency of Slovak Republic: VEGA grant no. 1/0120/23 “Environmental models as a tool for ecological and economic decisions making”.

Literature

1. Chatzilygeroudis, K., Hatzilygeroudis, I. & Perikos, I. (2021). Machine Learning Basics.
2. Demyanov, S. (2015). Regularization Methods for Neural networks and Related Models.
3. *Intelligent Computing for Interactive System Design: Statistics, Digital Signal Processing, and Machine Learning in Practice (1st ed.).*
4. Glorennec, P. Y. (2000). Reinforcement Learning: an Overview. Retrieved 22.6.2024 from: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b373b0c6e3b4fef4ac0534965708fc382343f8dc.
5. Grudniewicz, J. & Ślepaczuk, R. (2023). Application of machine learning in algorithmic investment strategies on global stock markets. Research in International Business and Finance.
6. Hutchinson, J. M., Lo, A. W., & Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks (NBER Articleing Paper No. w4718). National Bureau of Economic Research. https://ssrn.com/abstract=236673
7. Lee, T. & Cho, J. & Kwon, D. & Sohn, S. (2018). Global stock market investment strategies based on financial netarticle indicators using machine learning techniques. Expert Systems with Applications.
8. Li, J & Wang X. & Ahmad, S. & Huang, X. & Khan, Y. A. (2023), Optimization of investment strategies through machine learning, Heliyon.
9. Lo, A. W., & MacKinlay, A. C. (1994). A Nonparametric Approach to Pricing and Hedging Derivative Securities. *The Journal of Finance, 49*(5), 1815-1839. https://doi.org/10.1111/j.1540-6261.1994.tb00081.x
10. Priel, R. & Rokach, L. (2024). Machine learning-based stock picking using value investing and quality features. Neural Computing and Applications.
11. Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development, 3*(3), 210-229. https://doi.org/10.1147/rd.33.0210
12. Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Netarticle (RNN) and Long Short-Term Memory (LSTM) netarticle. *Physica D: Nonlinear Phenomena*.
13. Turing, A. M. (1950). Computing machinery and intelligence. *Mind, 59*(236), 433-460. https://doi.org/10.1093/mind/LIX.236.433
14. White, H. (1988). Economic prediction using neural networks: The case of IBM daily stock returns. *1988 IEEE International Conference on Neural networks* (Vol. 2, pp. 451-458). IEEE. https://doi.org/10.1109/ICNN.1988.23959