Vorhersage von Aktienwerten mit maschinellem Lernen

Studienarbeit

von
Luca Burghard

Bearbeitungszeitraum:	25.03.2024 - 07.06.2024
Immatrikulationsnummer:	9209388
Kurs:	TMT21B2
Studiengang:	Mechatronik
Hochschule:	Duale Hochschule Baden Württemberg Karlsruhe
Abgabetermin:	07.06.2024
Prüfer:	Steffen Quadt

Selbstständigkeitserklärung

Hiermit bestätige ich, dass ich meine Studienarbeit: “Vorhersage von Aktienwerten mit maschinellem Lernen“ selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere ferner, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

_____ _____

Ort, Datum Unterschrift

Hilfsmittel

Dieses Dokument wurde in LaTeX verfasst. Ausformulierungen wurden mit unterstützung von Sprachmodellen wie Github Copilot und OpenAI GPT generiert. Zum Überprüfen der Rechtschreibung und Grammatik wurde die Software LanguageTool verwendet. Sätze wurden zur besseren Verständlichkeit mit DeepL Write umgeschrieben. Übersetzungen wurden mit DeepL Translator erstellt.

Kurzfassung

Diese Arbeit behandelt die praktische Anwendung von Machine Learning Modellen zur Vorhersage von Aktienkursen, speziell am Beispiel von Adobe Inc. Die Untersuchung basiert auf Daten, die täglich von einem Raspberry Pi gesammelt und per API-Anfragen von verschiedenen Providern bezogen werden. Der Untersuchungszeitraum erstreckt sich vom 07.11.2023 bis zum 15.04.2024, und die gesammelten Daten umfassen insgesamt 161 Tage, die in einer CSV-Datei gespeichert wurden.

Der Kern der Arbeit besteht aus der Implementierung eines Rahmenprogramms, das einen standardisierten Ablauf für das Laden, Verarbeiten und Trainieren der Machine Learning Modelle bietet. Innerhalb dieses Rahmens werden verschiedene Modelle, wie Lineare Regression und KNN, unter Verwendung von Techniken wie GridSearch und RandomizedSearchCV auf ihre Effektivität geprüft.

Die Modelle werden dann auf die aufbereiteten Daten angewendet, wobei Schritte wie das Normalisieren der Daten, die Berechnung von Sentiment Scores aus Nachrichtentexten und das Hinzufügen weiterer Features durchgeführt werden. Die Ergebnisse des Modelltrainings werden mittels Mean Squared Error und R2 Score evaluiert und verglichen. Dabei wird untersucht, wie unterschiedliche Datenverarbeitungsschritte die Vorhersagegenauigkeit beeinflussen.

Abschließend wird ein Modell ausgewählt, das am besten performt, um die tägliche Preisveränderung von Adobe-Aktien vorherzusagen. Es wird eine Methode entwickelt, um aus den Vorhersagen Handelssignale zu generieren. Dazu wird Optuna verwendet, um optimale Schwellwerte für Kauf- und Verkaufssignale zu bestimmen, basierend auf simulierten Handelsstrategien, die darauf abzielen, den Depotwert zu maximieren. Die Ergebnisse zeigen, dass bestimmte Modellkonfigurationen und Schwellwerte das Potenzial bieten, profitable Handelssignale zu generieren.

Die Arbeit schließt mit einer Diskussion der Ergebnisse und der Darstellung der Kauf- und Verkaufssignale über die Zeit. Dies bietet eine visuelle Repräsentation der Leistung der Modelle und der Effektivität der Handelsstrategie im Untersuchungszeitraum.

Abstract

This thesis deals with the practical application of machine learning models to predict stock prices, specifically using the example of Adobe Inc. The study is based on data collected daily from a Raspberry Pi and obtained via API requests from various providers. The study period extends from 07.11.2023 to 15.04.2024, and the collected data comprises a total of 161 data points stored in a CSV file.

In the main part of the work, the data is first obtained and analyzed. This includes loading the data into a DataFrame, outputting the first lines to check the data structure, analyzing for missing values and checking the data types. The data is made up of various data formats, including numerical values with and without decimal places, as well as news about the share and its CEO.

The core of the work consists of implementing a framework that provides a standardized flow for loading, processing and training the machine learning models. Within this framework, different models, such as Linear Regression and KNN, are tested for their effectiveness using techniques such as GridSearch and RandomizedSearchCV. Focus is on optimizing the hyperparameters of the models.

The models are then applied to the processed data, performing steps such as normalizing the data, calculating sentiment scores from message texts and adding other relevant features. The results of the model training are evaluated and compared using Mean Squared Error and R2 Score. It is investigated how different data processing steps influence the prediction accuracy.

Finally, a model is selected that performs best to predict the daily price change of Adobe shares. A method is developed to generate trading signals from the predictions. Optuna is used to determine optimal thresholds for buy and sell signals based on simulated trading strategies that aim to maximize portfolio value. The results show that certain model configurations and thresholds offer the potenzial to generate profitable trading signals.

The paper concludes with a discussion of the results and the plotting of buy and sell signals over time. This provides a visual representation of the performance of the models and the effectiveness of the trading strategy over the study period.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

1 Einleitung

Im ersten Teil der Studienarbeit stand das Sammeln der Daten sowie die Literaturrecherche zu Machine Learning in Bezug auf den Aktienmarkt im Vordergrund. Dabei wurde darauf geachtet, geeignete Datenquellen zu finden und die relevanten Informationen zu extrahieren. Die gewonnenen Erkenntnisse aus der Literaturrecherche bilden die Grundlage für den weiteren Verlauf der Arbeit und dienen als Basis für die Implementierung eines Prototyps.

1.1 Zielsetzung

Ziel der arbeit ist es, ein Proof of Concept zu erhalten, dass eine Machine Learning unterstüzung beim Handeln von Aktien hilfreich ist und zu besseren Ergebnissen führt wie das Handeln nach eigener Rechere oder dem eigenen "Bauchgefühl".

1.2 Abgrenzung

Es ist nicht Ziel der Arbeit ein Marktfertiges Produkt zu entwickeln, sondern nur zu zeigen, dass es möglich ist mit Machine Learning bessere Ergebnisse zu erzielen als ohne.

2 Grundlagen

2.1 Machine-Learning-Modelle

In diesem Kapitel werden die verschiedenen Machine-Learning-Modelle beschrieben, die in dieser Studienarbeit verwendet werden. Es wird erläutert, wie jedes Modell funktioniert und in welchen Anwendungsbereichen es typischerweise zum Einsatz kommt.

2.1.1 Lineare Regression

Die lineare Regression ist ein statistisches Verfahren, das eine lineare Beziehung zwischen einer abhängigen Variable (y) und einer oder mehreren unabhängigen Variablen (X) modelliert. Das Modell findet häufig Anwendung in Vorhersageszenarien, beispielsweise zur Prognose von Preisen, Verkaufszahlen oder Temperaturwerten anhand vorhandener Daten.

Die lineare Beziehung wird durch die Gleichung $y(x) = mx + b$ beschrieben, wobei m die Steigung und b der y-Achsenabschnitt ist. Ziel ist es, die Parameter m und b so zu schätzen, dass sie die beste Anpassung an die Daten bieten.

Zur Bewertung der Modellgenauigkeit wird eine Verlustfunktion, wie der mittlere quadratische Fehler (MSE), herangezogen. Das Training des Modells erfolgt durch Anpassung der Parameter m und b, um den Verlust zu minimieren.

Abbildung 1 illustriert ein Beispiel für eine lineare Regression, bei der eine Gerade so durch die Datenpunkte gelegt wird, dass der Gesamtabstand zu diesen Punkten minimal ist. Wird ein neuer Wert x_{new} hinzugefügt, kann der zugehörige y-Wert (y_{new}) durch Einsetzen von x_{new} in die Gleichung prognostiziert werden.

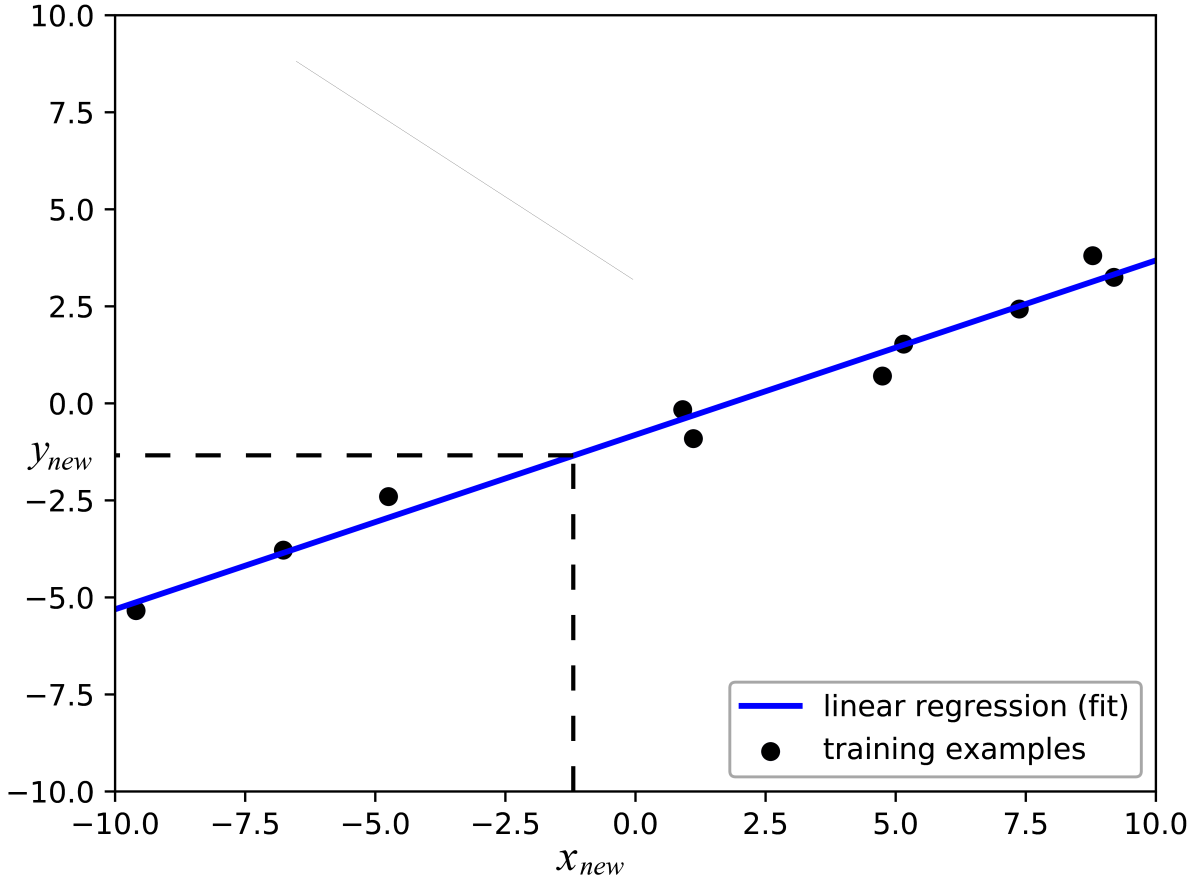

Abbildung 1: Lineare Regression (Quelle: `https://alioh.github.io/images/2019-10-25/ch3-fig1.png`, Zugriff 18.04.2024)

Lineare Modelle sind nicht nur auf eine Dimension beschränkt, sondern können auch in mehrdimensionalen Räumen angewandt werden, wo die Gleichung zu $y = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$ erweitert wird. Hierbei repräsentieren $w_1, w_2, \ldots, w_n$ die Gewichte der unabhängigen Variablen, was eine Modellierung komplexerer Zusammenhänge zwischen den Variablen ermöglicht. Für jeden zusätzlichen Parameter wird eine zusätzliche Dimension im Raum hinzugefügt, was die Modellkomplexität erhöht.

2.1.2 Entscheidungsbaum

Ein Entscheidungsbaum ist ein Modell, das Daten anhand eines baumartigen Graphen klassifiziert oder vorhersagt. Jeder Knoten im Baum entspricht einer Entscheidungsregel, und die Blätter des Baumes repräsentieren die Ergebnisse. Entscheidungsbäume finden häufig Anwendung in der Kreditbewertung und medizinischen Diagnostik.

Von einem Wurzelknoten ausgehend teilt sich der Baum in Zweige, die durch Entscheidungsregeln definiert sind. Überschreitet oder unterschreitet ein Merkmal einen bestimmten Schwellenwert, verzweigt sich der Baum entsprechend. Dieser Prozess

setzt sich rekursiv fort, bis ein Blatt erreicht wird, das das Endergebnis darstellt.

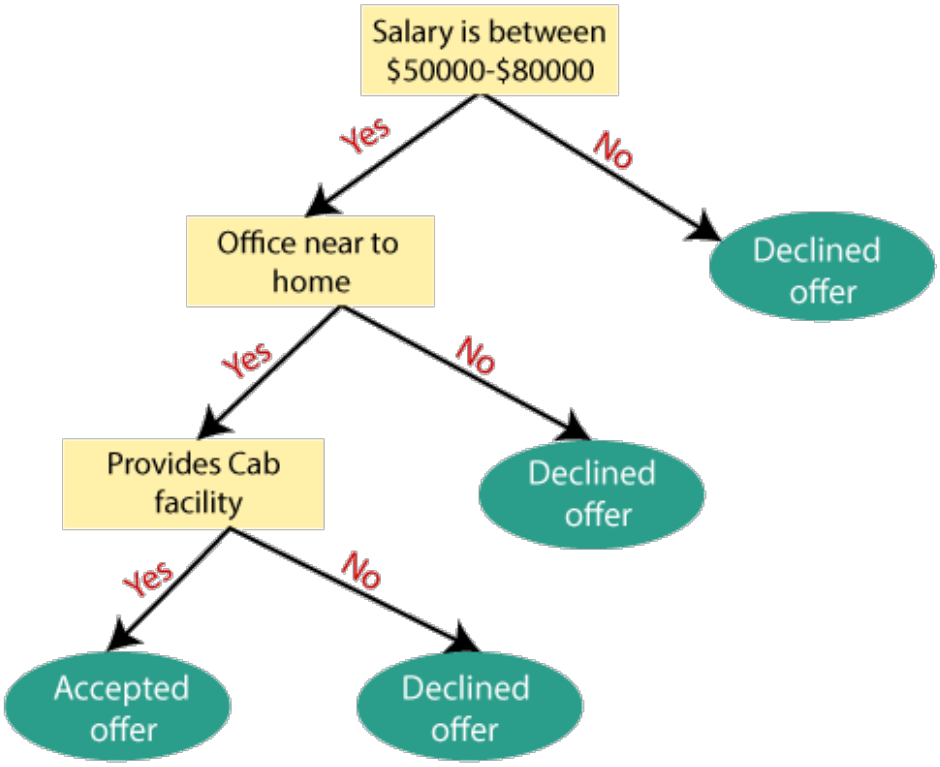

Abbildung 2: Entscheidungsbaum (Quelle: `https://static.javatpoint.com/tutorial/machine-learning/images/decision-tree-classification-algorithm2.png`, Zugriff 18.04.2024)

2.1.3 Random-Forest-Regressor

Der Random-Forest-Regressor verwendet eine Ensemble-Methode, die mehrere Entscheidungsbäume während des Trainingsprozesses erstellt und die Vorhersagen dieser Bäume mittelt. Diese Methode verbessert die Vorhersagegenauigkeit und reduziert die Gefahr der Überanpassung. Anwendung findet der Random-Forest-Regressor unter anderem bei der Vorhersage von Immobilienpreisen und Aktienmarktentwicklungen.

Wie bereits im Unterunterabschnitt 2.1.2 beschrieben, besteht ein Entscheidungsbaum aus mehreren Entscheidungsregeln, die zu einem Endergebnis führen. Der Random-Forest-Regressor baut auf einer Vielzahl solcher Bäume auf, die unabhängig voneinander auf zufälligen Teilmengen des Datensatzes trainiert werden, um die Variabilität zu reduzieren. Die finale Vorhersage ergibt sich aus der Mittelung der einzelnen Baumvorhersagen.

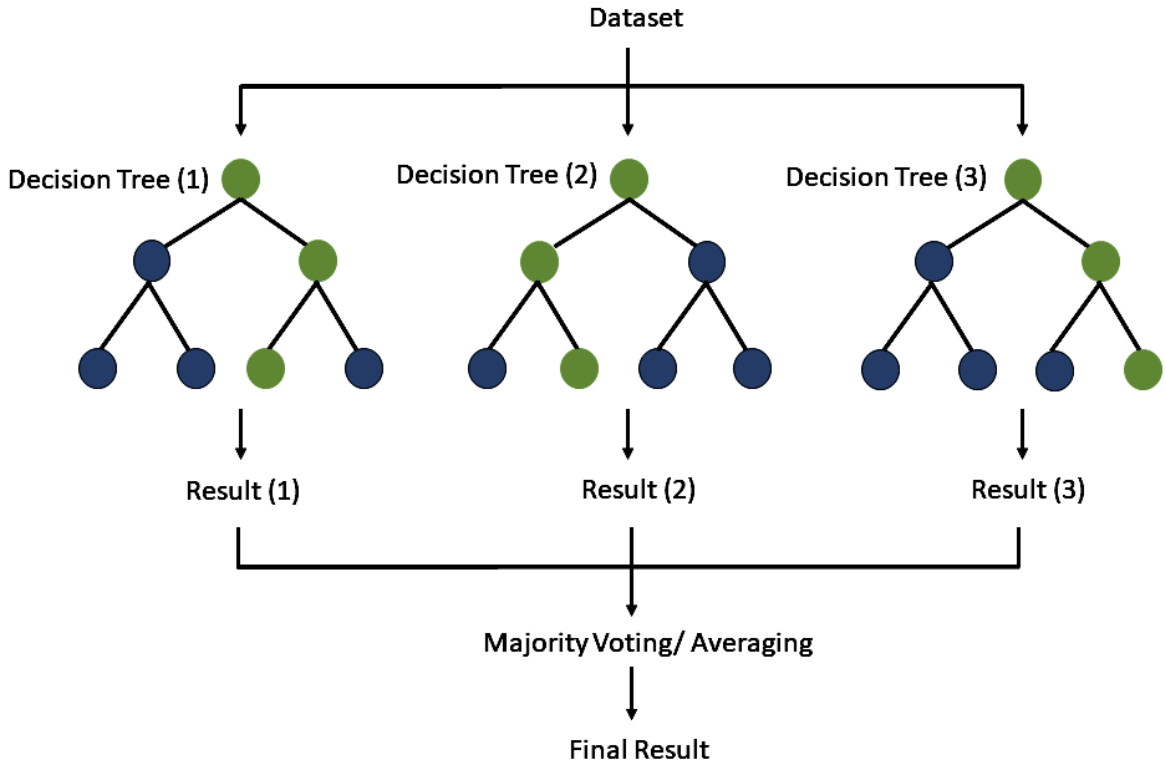

Abbildung 3: Random Forest (Quelle: `https://miro.medium.com/v2/resize:fit:720/format:webp/1*R3oJiyaQwyLUyLZL-scDpw.png`, Zugriff 18.04.2024)

Der Prozess der Vorhersage mit einem Random-Forest-Regressor wird in Abbildung 3 veranschaulicht, wo mehrere Entscheidungsbäume dargestellt sind, die unabhängig voneinander trainiert werden. Die Gesamtvorhersage resultiert aus der Durchschnittsbildung der Vorhersagen dieser Bäume.

2.1.4 Gradient-Boosting-Regressor

Das Gradient-Boosting ist eine weitere fortgeschrittenere Ensemble-Technik, die sequenziell Modelle aufbaut, wobei jedes nachfolgende Modell versucht, die Fehler des vorherigen Modells zu korrigieren. Häufig wird dieses Verfahren zur Erstellung robuster Vorhersagemodelle in der Finanzwirtschaft und Biotechnologie eingesetzt.

Im Gegensatz zum Random Forest, bei dem die Bäume unabhängig voneinander trainiert werden, erfolgt das Training der Bäume beim Gradient Boosting sequenziell. Das anfängliche Modell, das die Daten schlecht vorhersagt, wird kontinuierlich durch Hinzufügen neuer Modelle verbessert, die jeweils die verbleibenden Fehler korrigieren. Dieser Prozess wird so lange fortgesetzt, bis ein vordefiniertes Kriterium erreicht ist.

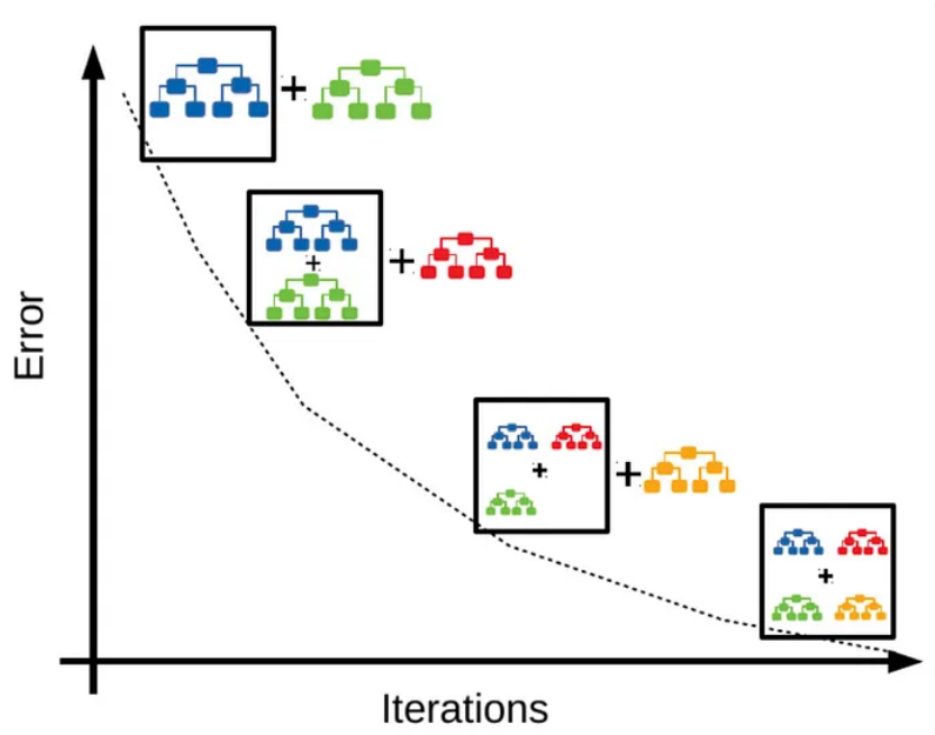

Abbildung 4: Gradient Boosting (Quelle: `https://miro.medium.com/v2/resize:fit:720/format:webp/1*OZPOQUKiaVmZOEMm_-8iYA.png`, Zugriff 18.04.2024)

Abbildung 4 zeigt, wie der Fehler in jedem Schritt der Iteration durch das Hinzufügen eines neuen Modells, das die Fehler des vorherigen Modells korrigiert, reduziert wird. Die Berechnung wird dadurch sehr rechenintensiv, aber die Genauigkeit der Vorhersagen wird verbessert.

2.1.5 K-Nearest Neighbors Regressor

Der K-Nearest Neighbors (KNN) Regressor basiert auf der Annahme, dass ähnliche Datenpunkte ähnliche Ergebnisse liefern. Dieses Modell wird zur Vorhersage von Werten neuer Datenpunkte eingesetzt und findet Anwendung in Bereichen wie Bilderkennung, Sprachverarbeitung und Finanzanalyse. Bei der Anwendung des KNN-Modells ist die Wahl der Anzahl von k Nachbarn sowie der Distanzmetrik entscheidend, da eine zu geringe Anzahl an Nachbarn zu einer Überanpassung und eine zu hohe Anzahl zu einer Unteranpassung führen kann.

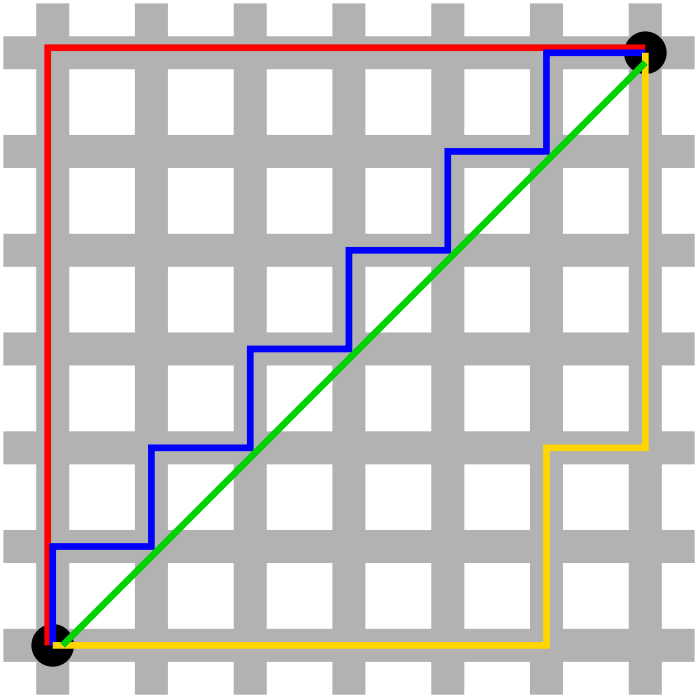

Abbildung 5: Manhatten (rot, blau, gelb) vs. Euklidischer (grün)Abstand (Quelle: `https://de.wikipedia.org/wiki/Manhattan-Metrik#/media/Datei:Manhattan_distance.svg`, Zugriff 30.04.2024)

Der Prozess des KNN-Modells umfasst drei Schritte:

1. Auswahl der Anzahl k, der zu betrachtenden nächsten Nachbarn sowie Festlegung einer geeigneten Metrik zur Berechnung der Distanz zwischen den Datenpunkten, beispielsweise der euklidische oder der Manhattan-Abstand wie in Abbildung 5 zu sehen ist.
2. Ermittlung der k nächstgelegenen Nachbarn des zu klassifizierenden Datenpunkts durch Berechnung der Distanzen zu allen anderen Datenpunkten. Die k Datenpunkte mit der geringsten Distanz werden ausgewählt.
3. Zuweisung einer Klassenbezeichnung zum zu klassifizierenden Datenpunkt durch eine Mehrheitsabstimmung unter den k nächsten Nachbarn. Die am häufigsten vorkommende Klasse unter den k Nachbarn wird als Vorhersage für den zu klassifizierenden Datenpunkt verwendet.

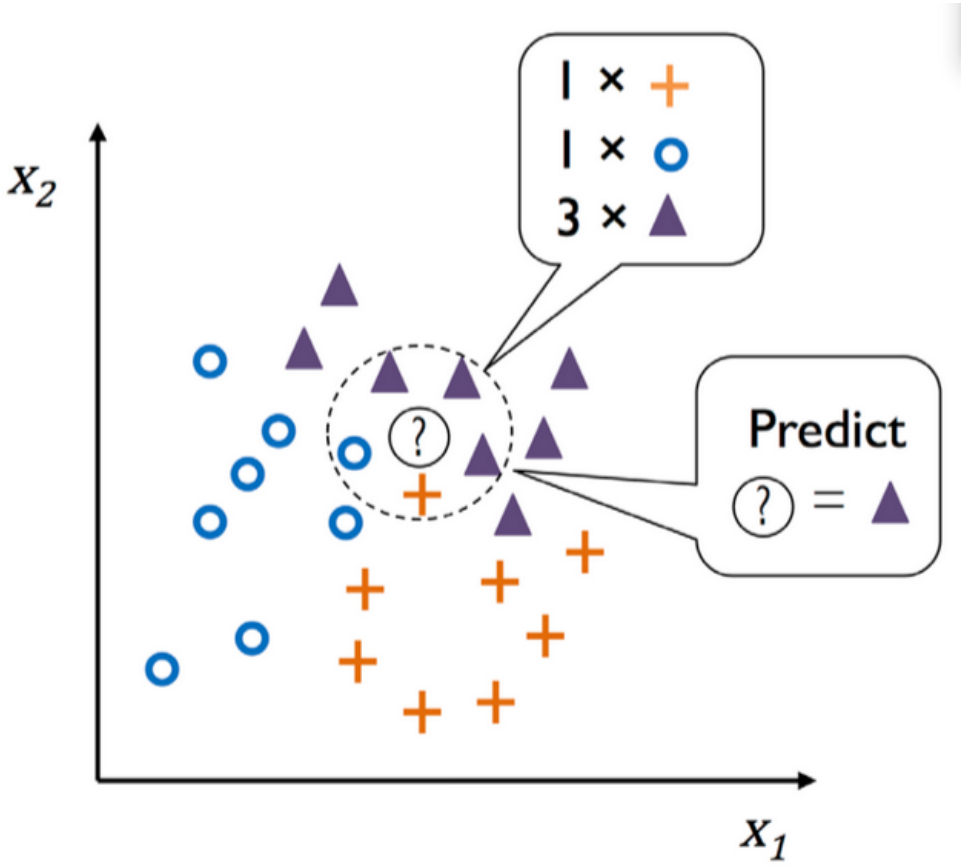

Abbildung 6: K-Nearest Neighbors (Quelle: Raschka u. a. 2022, Seite 99)

Abbildung 6 veranschaulicht ein Beispiel für den K-Nearest Neighbors-Ansatz, bei dem die Klassenzugehörigkeit eines Datenpunkts basierend auf den k nächstgelegenen Nachbarn bestimmt wird. In diesem Fall ist $k = 5$, und die Klassenzugehörigkeit des Datenpunkts wird durch eine Mehrheitsentscheidung der fünf nächsten Nachbarn bestimmt, wobei das lilafarbene Dreieck am häufigsten vorkommt und daher als Vorhersage für den zu klassifizierenden Datenpunkt verwendet wird.

2.1.6 Neuronales Netzwerk Regressor

Der neuronale Netzwerk-Regressor ist ein Modell, das auf der Struktur des menschlichen Gehirns basiert und künstliche Neuronen verwendet, um komplexe Beziehungen zwischen Variablen zu modellieren. Einsatzgebiete dieses Modells umfassen die Bilderkennung, Sprachverarbeitung und Finanzanalyse.

Ein künstliches neuronales Netzwerk besteht aus mehreren Schichten von Neuronen, die miteinander verbunden sind. Jedes Neuron erhält Eingaben von anderen Neuronen, führt eine Berechnung durch und gibt das Ergebnis weiter. Die Neuronen sind in Schichten organisiert: Die Eingabeschicht nimmt die Eingangsdaten auf, während die Ausgabeschicht die Vorhersage liefert und die verborgenen Schichten die komplexen

Beziehungen zwischen den Variablen modellieren.

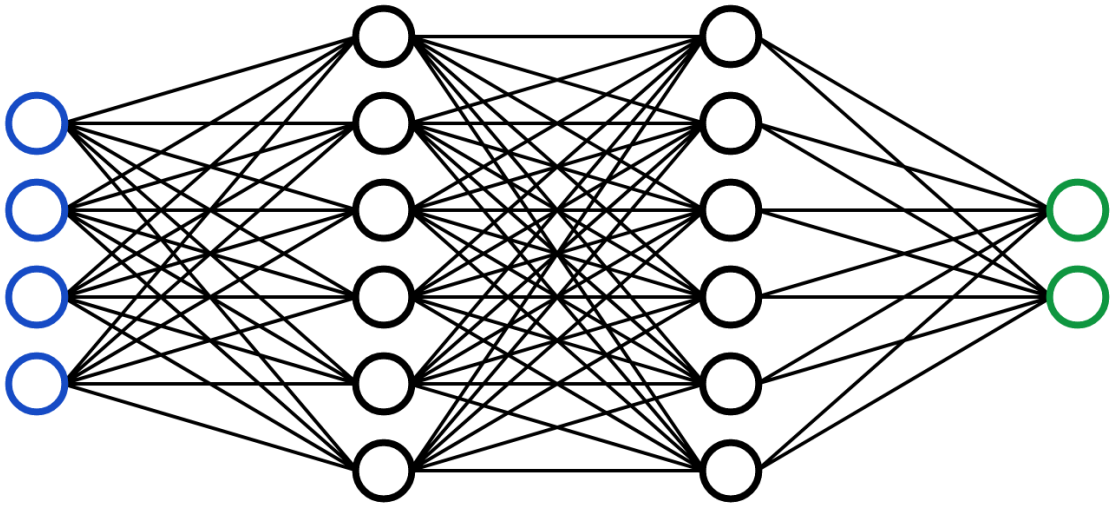

Abbildung 7: Neuronales Netzwerk (Quelle: `https://victorzhou.com/media/nn-series/network.svg`, Zugriff 18.04.2024)

Abbildung 7 zeigt ein Beispiel für ein künstliches neuronales Netzwerk mit einer Eingabeschicht, zwei verborgenen Schichten und einer Ausgabeschicht. Die Eingabeschicht nimmt die Daten auf, die verborgenen Schichten verarbeiten die Daten, indem sie die komplexen Beziehungen zwischen den Variablen modellieren, und die Ausgabeschicht liefert die Vorhersage des Modells.

2.1.7 Support Vector Machine Regressor

Die Support Vector Machine (SVM) ist ein Regressions- und Klassifikationsmodell, das darauf abzielt, eine Hyperplane zu finden, welche die größtmögliche Trennung zwischen zwei Klassen von Datenpunkten erreicht. Dieses Modell wird in der Bilderkennung, Textklassifizierung und Finanzanalyse eingesetzt.

Die SVM arbeitet, indem sie eine Trennlinie (oder Hyperplane) zwischen zwei Klassen von Datenpunkten findet, die den größtmöglichen Abstand zwischen den Klassen aufweist. Die SVM maximiert diesen Abstand, um die Genauigkeit der Klassifizierung zu verbessern, und kann sowohl für lineare als auch für nicht-lineare Daten eingesetzt werden, was sie von der linearen Regression unterscheidet.

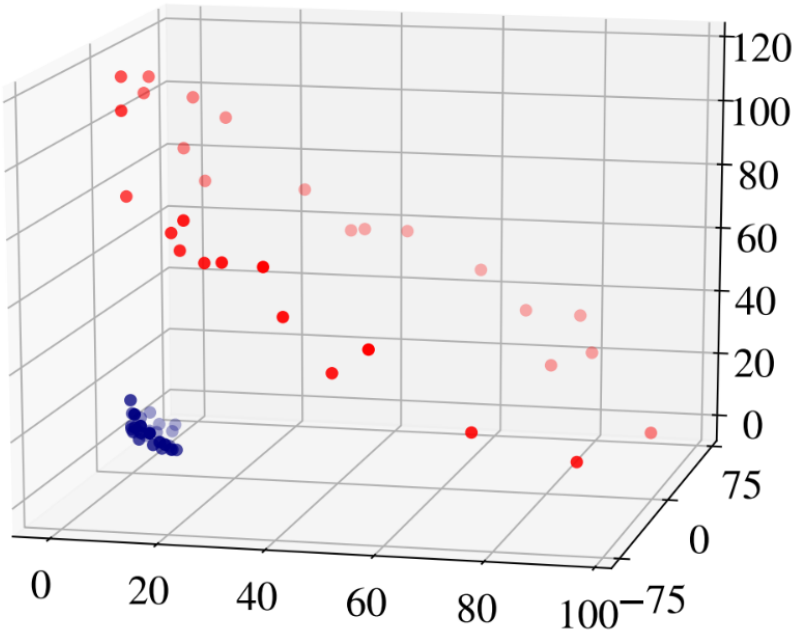

Abbildung 8: Support Vector Machine (Quelle: vgl. Burkov 2019, Kapitel: Fundamental Algorithms, Seite 12-14)

Wie in Abbildung 8 dargestellt, ermöglicht die SVM die Trennung von Daten, die im zweidimensionalen Raum nicht separierbar sind, indem sie in einen höherdimensionalen Raum projiziert werden, in dem eine lineare Trennung möglich ist.

2.2 Benchmarks

Benchmarks werden genutzt, um ähnliche Produkte oder Software miteinander zu vergleichen. Bei Computerprogrammen wird oft die Laufzeit oder die Genauigkeit der Vorhersagen gemessen. In dieser Arbeit werden Benchmarks eingesetzt, um die Genauigkeit der Modelle zu bewerten.

2.2.1 Mean Squared Error (MSE)

Der Mean Squared Error (MSE) ist der am häufigsten verwendete Benchmark zur Bewertung von Machine-Learning-Modellen. Die Formel für den MSE lautet:

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

$$n = \textit{Anzahl der Datenpunkte} \tag{2}$$

$$y_i = \textit{Beobachtete Werte} \tag{3}$$

$$\hat{y}_i = \textit{Vorhergesagte Werte} \tag{4}$$

2.2.2 R2-Score

Der R2-Score misst die Genauigkeit eines Modells im Vergleich zu einem einfachen Durchschnittsmodell. Die Formel hierzu lautet:

$$\textit{R2-Score} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5}$$

Der R2-Score variiert zwischen 0 und 1, wobei ein Wert von 1 eine perfekte Vorhersage und ein Wert von 0 keine Vorhersagekraft des Modells bedeutet.

2.2.3 Vergleichsstrategie in diesem Projekt

Für den Vergleich der Modelle wird eine umfassende Tabelle erstellt, in der jede Spalte einem Modell entspricht. In den Zeilen stehen verschiedene Konfigurationen oder Datenverarbeitungsschritte. In den Zellen befinden sich die Benchmarks (Mean Squared Error und R2-Score). Am Ende der Analyse können die Zellen farblich markiert sowie Durchschnitte berechnet werden, um festzustellen, welches Modell insgesamt am besten abschneidet.

2.3 Datenverarbeitung

Die Datenverarbeitung bezieht sich auf die vorbereitenden Schritte, die notwendig sind, um Rohdaten in ein Format zu überführen, das effektiv von Datenanalysesystemen oder Machine-Learning-Algorithmen verarbeitet werden kann. Dieser Prozess ist entscheidend, da er die Qualität und Effektivität der Datenanalyse und Modellvorhersagen verbessert.

Im Rahmen dieses Projekts wurden kontinuierlich weitere Datenverarbeitungsschritte integriert, um die Datenqualität zu erhöhen. Zu diesen Schritten gehören unter anderem das Entfernen von fehlenden Werten, das Skalieren der Daten, das Kodieren von kategorialen Variablen und das Aufteilen der Daten in Trainings- und Testsets.

2.4 Hyperparameter-Tuning

Hyperparameter-Tuning ist der Prozess der Optimierung der Parameter, die die Konfiguration eines Machine-Learning-Modells steuern. Ziel ist es, die beste Kombination

von Parametern zu finden, die die Leistung des Modells maximiert. Zu den Methoden des Hyperparameter-Tunings gehören unter anderem Optuna, eine Bibliothek für automatische Hyperparameter-Optimierung, und Randomized Grid Search, eine Technik, die eine zufällige Auswahl von Parametern aus einer definierten Liste testet, um effizient optimale Modelleinstellungen zu identifizieren.

2.4.1 Optuna

Optuna ist eine Bibliothek für automatische Hyperparameter-Optimierung, die Bayesian Optimization verwendet, um die besten Hyperparameter für ein Modell zu finden. In diesem Projekt wird Optuna genutzt, um den optimalen Schwellenwert für Kauf- oder Verkaufsentscheidungen zu bestimmen.

2.4.2 Randomized Grid Search

Randomized Grid Search ist eine Technik, die eine zufällige Auswahl von Parametern aus einer definierten Liste testet, um effizient die besten Hyperparameter für ein Modell zu finden. In diesem Projekt wurde eine JSON-Datei erstellt, die alle zu testenden Parameter für die in Machine-Learning-Modelle aufgeführten Modelle enthält. Die Bibliothek führt dann eine Reihe von Tests durch und gibt die optimalen Parameter zurück.
Abbildung 9 zeigt eine Visualisierung des Randomized Grid Search-Prozesses. In dem vereinfachten Fall werden nur zwei Parameter getestet, um die optimale Kombination zu finden. Der grüne Punkt zeigt die optimale Kombination der Parameter, die den besten Score erzielt hat.

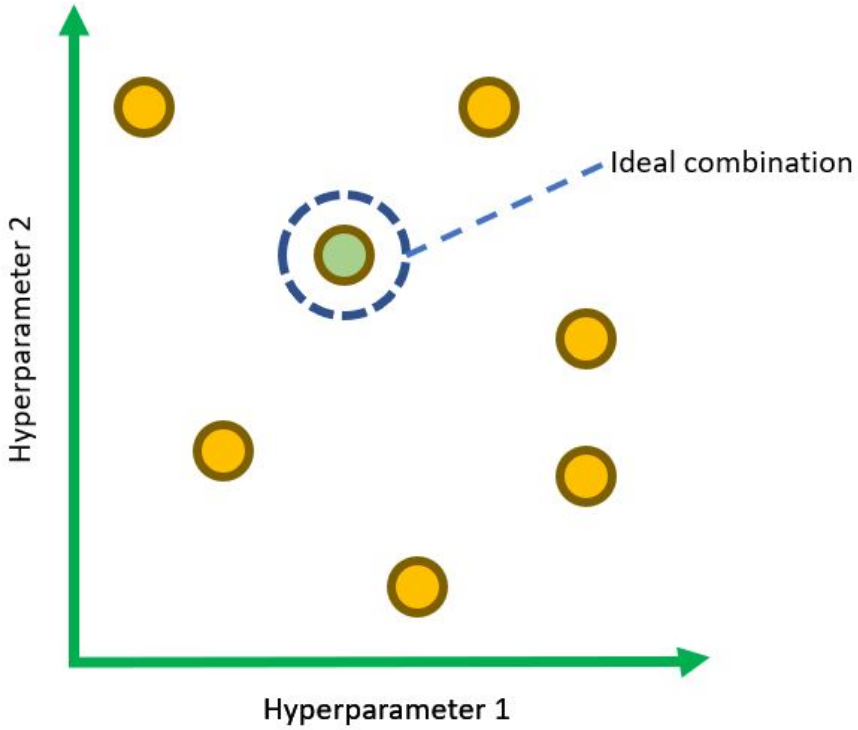

Abbildung 9: Randomized Grid Search (Quelle: `https://editor.analyticsvidhya.com/uploads/72089randomized.JPG`, Zugriff 30.04.2024)

2.5 Methodik

In einem Programmierprojekt ist es wesentlich, nicht nur den Code zu schreiben, sondern auch eine Methode zu haben, um die Ergebnisse zu überprüfen und zu vergleichen.

Wie in Benchmarks erläutert, werden zwei Benchmarks verwendet, um die Modelle zu vergleichen: der Mean Squared Error und der R2-Score. Aus diesen beiden Benchmarks wird ein Gesamtscore berechnet, indem der R2-Score invertiert wird, sodass ein kleinerer Wert besser ist als ein größerer. Dazu werden alle Werte von 1 abgezogen, sodass ein Score von 0.7 zu einem Score von 0.3 wird.

Im nächsten Schritt werden der Mean Squared Error und der invertierte R2-Score multipliziert, um den Gesamtscore zu erhalten. Die Formel für den Gesamtscore lautet:

$$Gesamtscore = (1 - R2) \cdot MSE \tag{6}$$

Die Vorhersagequalität wird iterativ verbessert, indem einzelne Datenverarbeitungsschritte aufeinander aufbauen. Diese Schritte werden anhand des Gesamtscores gemessen. Dieser wird zusammen mit einer Beschreibung der durchgeführten Änderun-

gen in einer Tabelle festgehalten. Die Tabelle enthält alle Modelle und die Scores für die verschiedenen Schritte. Am Ende wird der Durchschnitt der Scores berechnet, um zu ermitteln, welches Modell insgesamt am besten abgeschnitten hat. Des Weiteren lässt sich so auch feststellen, welche Schritte die größte Verbesserung gebracht haben und wie die einzelnen Modelle auf die Schritte reagiert haben.

3 Hauptteil

Der Hauptteil der Arbeit befasst sich mit der praktischen Umsetzung der Theorie. Der Code wird in Python implementiert und die verschiedenen Machine Learning Modelle werden auf die vorliegenden Daten angewendet. Die Modelle werden trainiert, getestet und die Ergebnisse werden ausgewertet und miteinander verglichen.

3.1 Datenbeschaffung

Die Daten wurden mithilfe eines Python-Skripts gesammelt, das täglich auf einem Raspberry Pi ausgeführt wird. Das Raspberry Pi sendet API-Anfragen an mehrere Datenanbieter und sammelt Informationen über die ausgewählte Aktie Adobe Inc., wie im ersten Teil beschrieben. Das Startdatum ist der 07.11.2023 und das Enddatum ist der 15.04.2024, um die Modelle vergleichbar zu machen. Die gesammelten Daten werden in einer CSV-Datei gespeichert. Insgesamt stehen 161 Datenpunkte zur Verfügung, auf denen die Modelle trainiert werden können.

3.2 Datenanalyse

Bevor die Daten verarbeitet werden, ist es sinnvoll, eine Analyse durchzuführen und die Struktur der Daten zu verstehen. Die Daten werden in ein Dataframe geladen und die ersten 5 Zeilen werden ausgegeben. Es wird überprüft, ob fehlende Werte vorhanden sind und die Datentypen der einzelnen Spalten werden überprüft. Die erste Zeile enthält das Datum des jeweiligen Tages, gefolgt von den Daten. Wenn die Daten von einem anderen Tag stammen, wird das Datum, an dem die Daten gesammelt wurden, angegeben.

Der Rohdatensatz enthält insgesamt 126 Spalten, darunter Daten in verschiedenen Formaten, Zahlen mit und ohne Nachkommastellen, Informationen über die Aktie und den CEO des Unternehmens und vieles mehr.

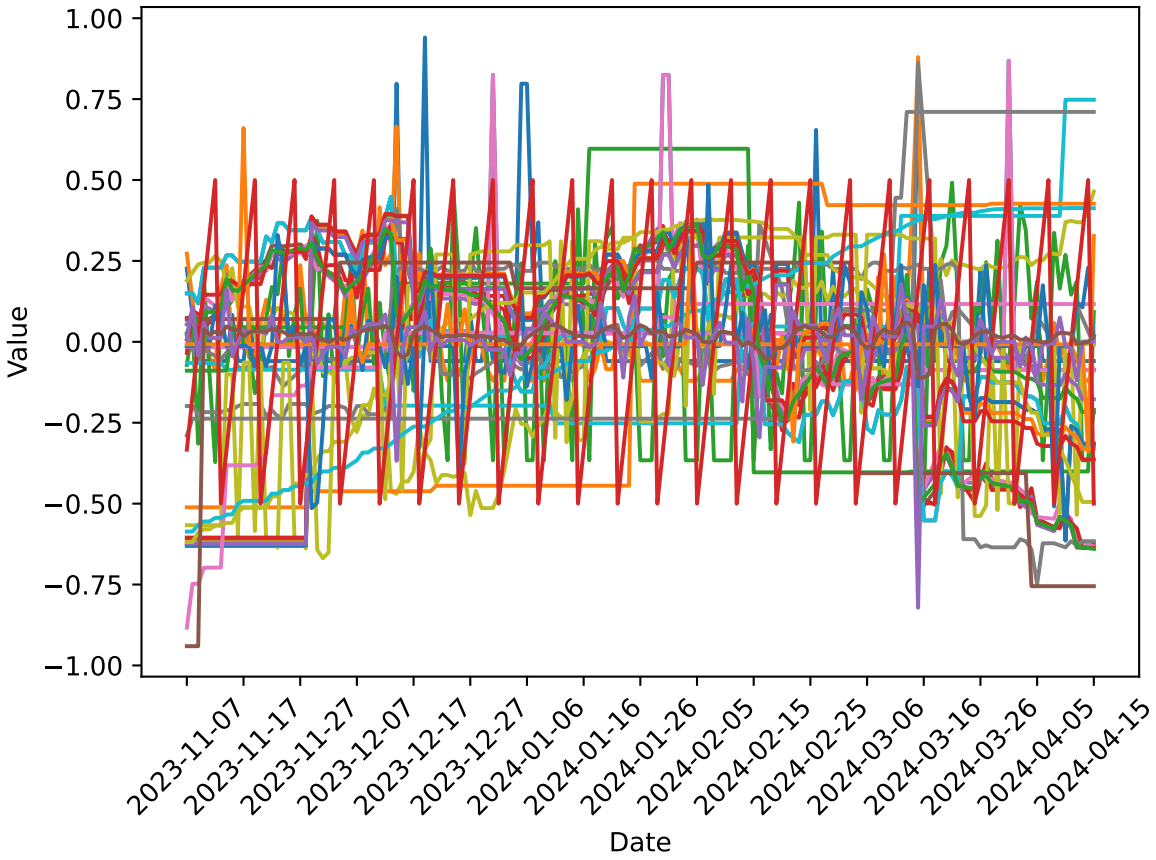

Abbildung 10: Features normalisiert

Die Features können nachdem sie normalisiert werden einzeln geplottet werden. Abbildung 10 zeigt alle Features in einem Plot, um die Verteilung der Daten zu visualisieren.

3.3 Rahmenprogramm

Wie im Abschnitt Methodik erläutert, wird ein „Rahmenprogramm“ implementiert, das den gesamten Prozess strukturiert. Innerhalb dieses Rahmens können weitere Schritte zur Datenverarbeitung durchgeführt werden und die Modelle können wiederholt auf die Daten angewendet werden.

Der erste Schritt im Rahmenprogramm besteht darin, eine Funktion namens „logger“ zu definieren. Die Logger Funktion funktioniert ähnlich wie das Print-Statement, speichert jedoch die Ausgabe in einer Datei. Der Logger wird in jedem Schritt des Programms verwendet, um die Ausgaben zu protokollieren. Dies ist wichtig, um die durchgeführten Schritte später nachvollziehen zu können, ohne das Programm erneut ausführen zu müssen. Der Aufruf des Loggers sieht wie folgt aus:

```
1 # Start des Programms, Zeitstempel wird protokolliert
2 logger("Das Programm wurde am " + str(datetime.now()) + "
    gestartet.")
```

Code 1: Logger Initialisierung

Das Rahmenprogramm beginnt mit dem Einlesen der Rohdaten und der Konvertierung der ersten Spalte mit dem Datum in ein Datumsformat. Anschließend können die Daten sortiert werden.

Es folgen unvermeidliche Schritte, die erforderlich sind, um die Daten für die Anwendung der Machine Learning Modelle vorzubereiten. Das Rahmenprogramm soll sicherstellen, dass die Modelle trainiert werden können. Weitere Features können später hinzugefügt werden.

Schritt	Beschreibung
1	Daten einlesen in einen Dataframe
2	Daten sortieren nach dem Datum
3	Platz für weitere Schritte, die in Tabelle 3 erklärt werden.
4	Alle nicht-numerischen Daten, wie Nachrichten entfernen
5	Leere Zellen füllen durch lineare Interpolation
6	Spalte erstellen, welche die Veränderung des Aktienwertes zum nächsten Tag in Prozent enthält (Target)
7	Daten in Trainings- und Testdaten aufteilen
8	Modelle trainieren und die Hyperparameter mit Randomized Search CV finden
9	Berechnung des Mean Squared Error und R2 Score für jedes Modell
10	Ergebnisse in einer Datei speichern
11	Ergebnisse analysieren und auswerten
12	Verschiedene Plots erstellen, um die Ergebnisse zu visualisieren.

Tabelle 1: Schritte des Rahmens

Das Programm wird nach jeder Änderung, z.B. dem Hinzufügen eines Features, erneut ausgeführt und der Rahmen wird durchlaufen. Dadurch kann die Auswirkung der einzelnen Schritte, die an dritter Stelle eingefügt werden auf die Modelle genau betrachtet werden.

3.4 Training der Modelle

Die verschiedenen Modelle haben unterschiedliche Hyperparameter, die gefunden werden müssen. Eine gängige Methode ist die Verwendung eines Gridsearch, bei dem alle möglichen Kombinationen von Hyperparametern ausprobiert werden. Dies ist jedoch rechenintensiv und zeitaufwändig.

Ein alternativer Ansatz ist die Verwendung des RandomizedSearchCV, wie in Unterunterabschnitt 2.4.2 erläutert. Da das Programm auf einem handelsüblichen Laptop ausgeführt wird, wird diese Methode gewählt, da sie weniger rechenintensiv ist.

Die Modelle werden in einer Schleife trainiert und getestet und in einer Liste mit ihren Hyperparametern gespeichert. Die Ergebnisse werden in einer Datei gespeichert und können später ausgewertet werden.

Ein Teil der Liste sieht wie folgt aus:

```
    models = [
        {
            "name": "Lineare Regression",
            "model": LinearRegression(),
            "param_grid": {}
        },
        {
            "name": "KNN",
            "model": KNeighborsRegressor(),
            "param_grid": {
                'n_neighbors': [3, 5, 11, 19],
                'weights': ['uniform', 'distance'],
                'metric': ['euclidean', 'manhattan']
            }
        }
    ]
```

Code 2: Liste der Modelle

Der Listenauschnitt enthält die Lineare Regression und den KNN Regressor. Der KNN Regressor hat Hyperparameter, die getestet werden müssen, während die Lineare Regression keine Hyperparameter hat.

Der RandomizedSearchCV kann dann einfach wie folgt aufgerufen werden, um die besten Hyperparameter für das Modell zu finden:

```
    random_search = RandomizedSearchCV(model["model"], model["
       param_grid"], n_iter=5, cv=5, n_jobs=-1, verbose=2)
    random_search.fit(X_train, y_train)
```

Code 3: RandomizedSearchCV

Der RandomizedSearchCV wird für jedes Modell in der Liste aufgerufen, indem eine Schleife über die Liste läuft.

3.5 Datenverarbeitungsschritte

Die Datenverarbeitungsschritte sind die Schritte, die nachträglich in den Rahmen eingefügt werden. Nach jedem der unten aufgelisteten Schritte werden die Modelle erneut trainiert und getestet.

Die Tabelle zeigt die Datenverarbeitungsschritte als Index und den Wert des jeweiligen Modells. Der Wert in der Tabelle ist der Gesamtscore, der im Abschnitt Benchmarks eingeführt wurde.

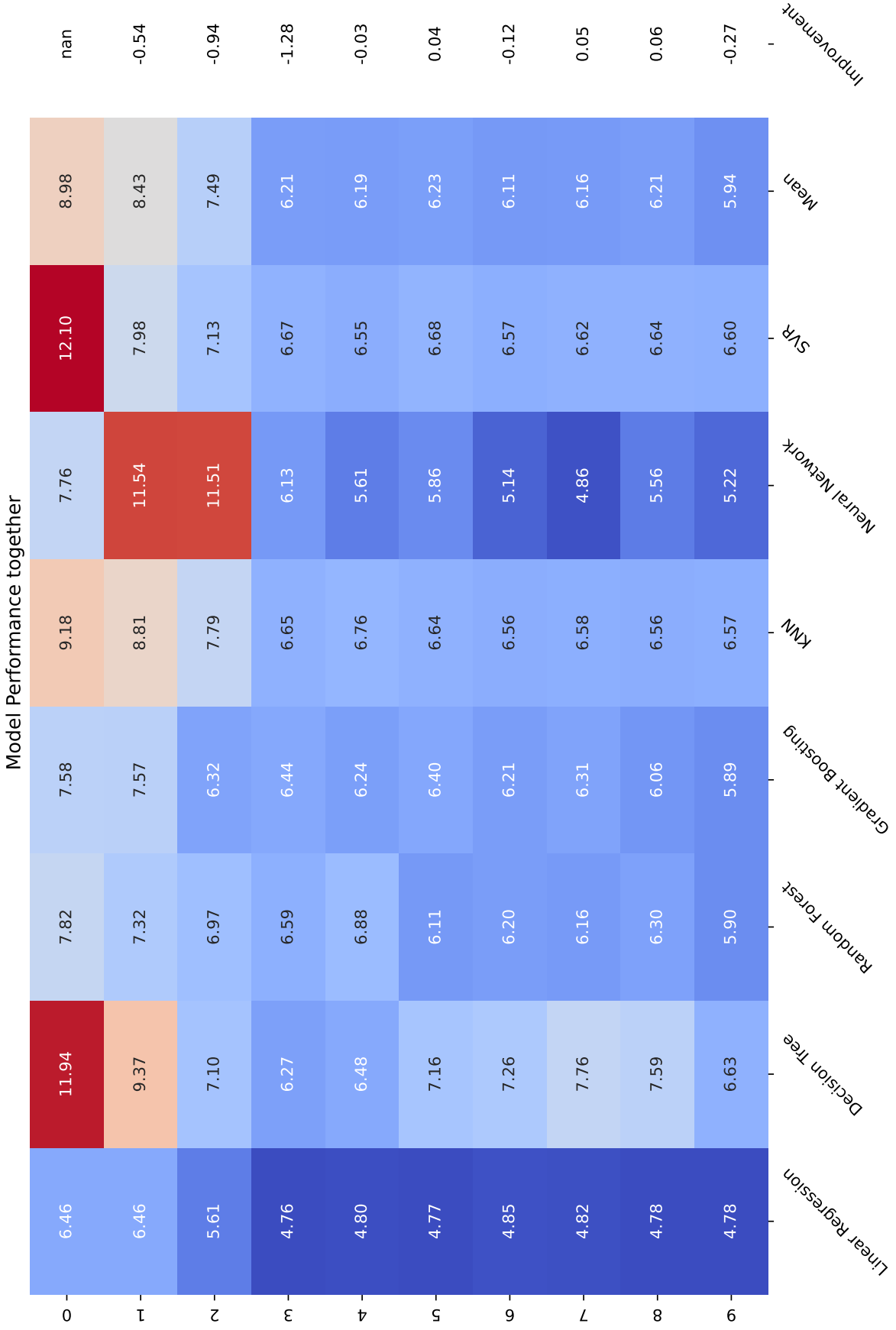

Tabelle 2: Gesamtscore

Schritt	Beschreibung
0	**Entfernung nicht-numerischer oder spärlicher Merkmale**: Nicht-numerische Merkmale sowie Merkmale mit weniger als fünf Werten wurden aus dem Datensatz entfernt, um die Qualität der Analyse zu verbessern.
1	**Datenstandardisierung mit Z-Score**: Die Daten wurden mittels Z-Score standardisiert, um die Mittelwerte auf 0 und die Standardabweichungen auf 1 zu normieren.
2	**Lineare Interpolation für Wochenendwerte**: Für fehlende Werte an Wochenenden wurde eine lineare Interpolation durchgeführt, um eine kontinuierliche Datensequenz zu gewährleisten.
3	**Berechnung des nächsten Tagespreises mit Durchschnittspreis**: Anstelle des Schlusspreises wurde der Durchschnittspreis verwendet, um die prozentuale Preisänderung des folgenden Tages zu berechnen.
4	**Wochentagkodierung von 0 bis 6**: Die Wochentage wurden als ganze Zahlen von 0 (Montag) bis 6 (Sonntag) kodiert, um eine eindeutige und effiziente Verarbeitung zu ermöglichen.
5	**Preisänderung im Vergleich zum Vortag**: Die Preisänderung im Vergleich zum Vortag wurde berechnet, um kurzfristige Markttrends analysieren zu können.
6	**Preisänderung im Vergleich zu drei Tagen zuvor**: Die Preisänderung im Vergleich zu drei Tagen zuvor wurde ermittelt, um mittelfristige Preisbewegungen zu erfassen.
7	**Sentimentanalyse mit Finbet**: Tägliche Sentiment-Scores wurden unter Verwendung von Finbet generiert, indem CEO- und Unternehmensnachrichten analysiert und in Stimmungswerte übersetzt wurden.
8	**Verwendung des Standard Scalers**: Anstelle des Z-Scores wurde der Standard Scaler aus sklearn für die Datenstandardisierung verwendet, um eine Alternative zur Z-Score-Methode zu bieten.
9	**PCA mit automatischer Dimensionsreduktion**: Die Hauptkomponentenanalyse (PCA) wurde angewandt, wobei die Anzahl der Komponenten automatisch basierend auf der erklärten Varianz ausgewählt wurde.

Tabelle 3: Erklärung der Schritte

Das beste Modell wird ausgewählt und verwendet, um die Preisänderung für den nächsten Tag vorherzusagen. Der folgende Graph zeigt die tatsächliche und die vorhergesagte Preisänderung.

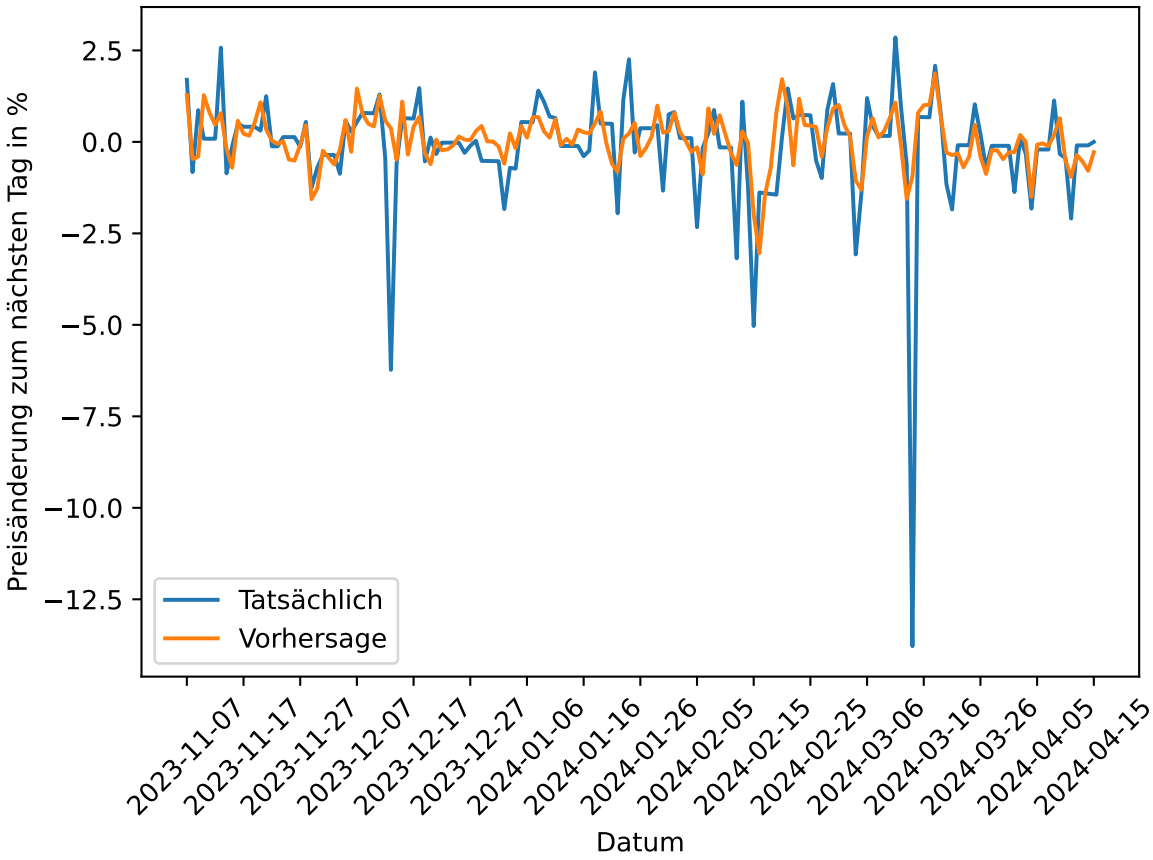

Abbildung 11: Vorhersage der Preisänderung

3.6 Anwendung

Die Modelle sagen die Preisänderung für den nächsten Tag voraus. Da die Nachrichten und Daten von den API-Providern in der kostenfreien Version nur mit einem Tag verzögerung verfügbar sind, muss bei der Anwendung auf die Uhrzeit geachtet werden. Die Berechnung und Vorhersage erfolgen morgens vor der Öffnung des Marktes um 07:00 Uhr.

Um eine Vorhersage zu treffen, wird das Modell mit dem besten Gesamtscore ausgewählt und auf alle verfügbaren Daten angewendet.

Es ist jedoch wichtig zu beachten, dass die Vorhersage eine Preisänderung in Prozent ist und kein Kauf- oder Verkaufssignal. Wie aus der prozentualen Preisänderung ein Handelssignal abgeleitet wird, wird im Abschnitt Vorhergesagte Prozente zu Handelssignalen erläutert.

3.7 Vorhergesagte Prozente zu Handelssignalen

Die Vorhersage gibt die prozentuale Preisänderung an. Diese Änderung kann positiv oder negativ sein. Um aus der Preisänderung ein Handelssignal abzuleiten, wird ein

Schwellenwert festgelegt. Wenn die Änderung größer als der Schwellenwert ist, wird ein Kaufsignal generiert. Wenn die Änderung kleiner als der Schwellenwert ist, wird ein Verkaufssignal generiert.

Dieser Schwellenwert muss optimiert werden. Hierfür wird das zuvor beschriebene Optuna verwendet. Optuna sucht den besten Schwellenwert innerhalb eines festgelegten Bereichs. Für ein Verkaufssignal wird ein Bereich von 0 % bis -10 % und für ein Kaufsignal ein Bereich von 0 % bis 10 % gewählt. Optuna benötigt jedoch eine Funktion, die optimiert werden kann. Hierfür wurde ein Simulationsprogramm entwickelt. Optuna versucht, den Depotwert zu maximieren, der sich aus dem Kauf und Verkauf der Aktie ergibt. Der Ablauf ist wie folgt:

1. Es gibt zwei Konten: das Depot und das Cash. Anfangs befinden sich 500 € im Depot und 500 € in Cash.
2. Optuna wählt zwei Schwellenwerte, einen für den Kauf und einen für den Verkauf.
3. Wenn die vorhergesagte Änderung größer als der Kaufschwellenwert ist, wird so viel von der Aktie gekauft, bis das Cash aufgebraucht ist. Ein Kauf kostet außerdem 1 € Gebühr.
4. Wenn die vorhergesagte Änderung kleiner als der Verkaufschwellenwert ist, wird 50 % des Depots verkauft. Ein Verkauf kostet ebenfalls 1 € Gebühr. Das Depot wird dann in Cash umgewandelt.

Optuna führt diesen Ablauf für eine festgelegte Anzahl von Iterationen durch und versucht, den Depotwert zu maximieren. Der Depotwert ist die Summe aus Cash und Depotwert.

Am Ende erhält man die besten Schwellenwerte für den Kauf und den Verkauf. Diese Schwellenwerte werden dann in der Anwendung verwendet, um aus der Vorhersage ein Handelssignal abzuleiten.Abbildung 12 zeigt die Ergebnisse von Optuna. In der Mitte ist der Buy Threshold und auf der rechten Seite der Sell Threshold. Links im Graph ist der Depotwert dargestellt, den es zu maximieren gilt. Die Linien zeigen die verbindungen zwischen den einzelnen Punkten, also bei welchen Parametern, welcher Depotwert erreicht wurde.

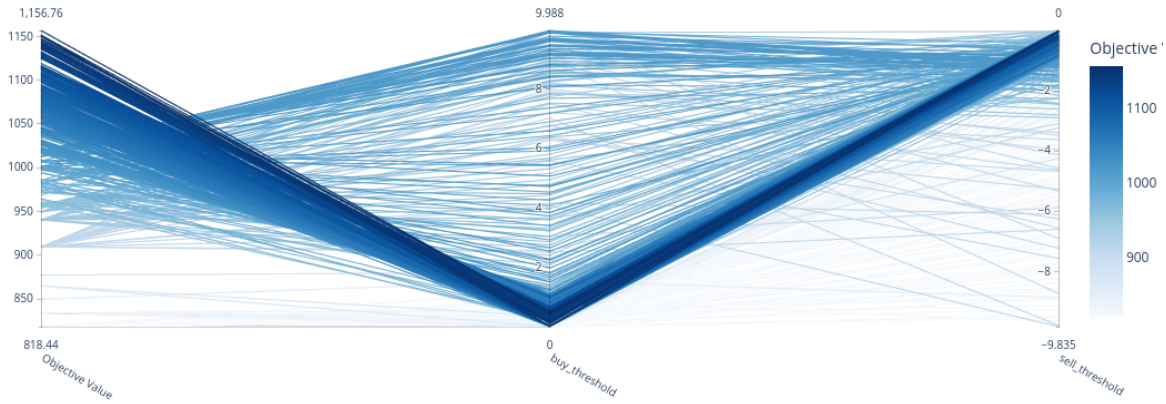

Abbildung 12: Optuna Parallel Plot

Des Weiteren lässt sich der Verlauf der Optimierung in einem Graphen darstellen. Abbildung 13 zeigt den Verlauf der Optimierung. Links ist wieder der Depotwert dargestellt, der maximiert werden soll. Die horizontale Achse zeigt die Anzahl der Iterationen. Die Punkte sind die Versuche und die rote Linie zeigt den Verlauf der besten Werte. Die besten Werte sind die, die den Depotwert maximieren.

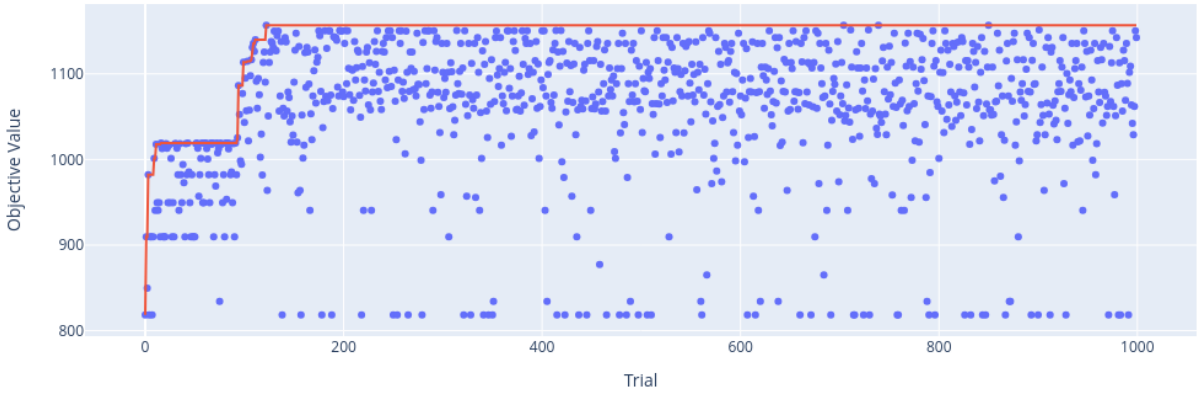

Abbildung 13: Optuna Verlauf

3.8 Ergebnisse

Die Ergebnisse sind zur übersichtlichen Darstellung in einer Tabelle zusammengefasst.

Beschreibung	Ergebniss
Bestes Modell	Lineare Regression
Gesamtscore (siehe Unterabschnitt 2.2)	4.78
Schwellenwert Kauf	0,46 %
Schwellenwert Verkauf	-0,011 %
Erreichter Depotwert	1156,76 €
Erreichte Differenz zur Haltstrategie	33,93 %

Tabelle 4: Schritte des Rahmens

Mit den vorhergesagten Werten und den Schwellenwerten gibt es für jeden Tag ein Kauf-, Halt- oder Verkaufssignal. Diese Signale werden in einem Graphen dargestellt, wobei grüne oder rote Balken verwendet werden.

Bei einem Kauf wird das Cash in die Aktie investiert, bei einem Verkauf wird 50 % des Depots verkauft.

Der folgende Graph zeigt den Verlauf der Simulation, bei der die Signale berücksichtigt werden und der Aktienwert normalisiert wird, um eine vergleichbare Skalierung zu gewährleisten.

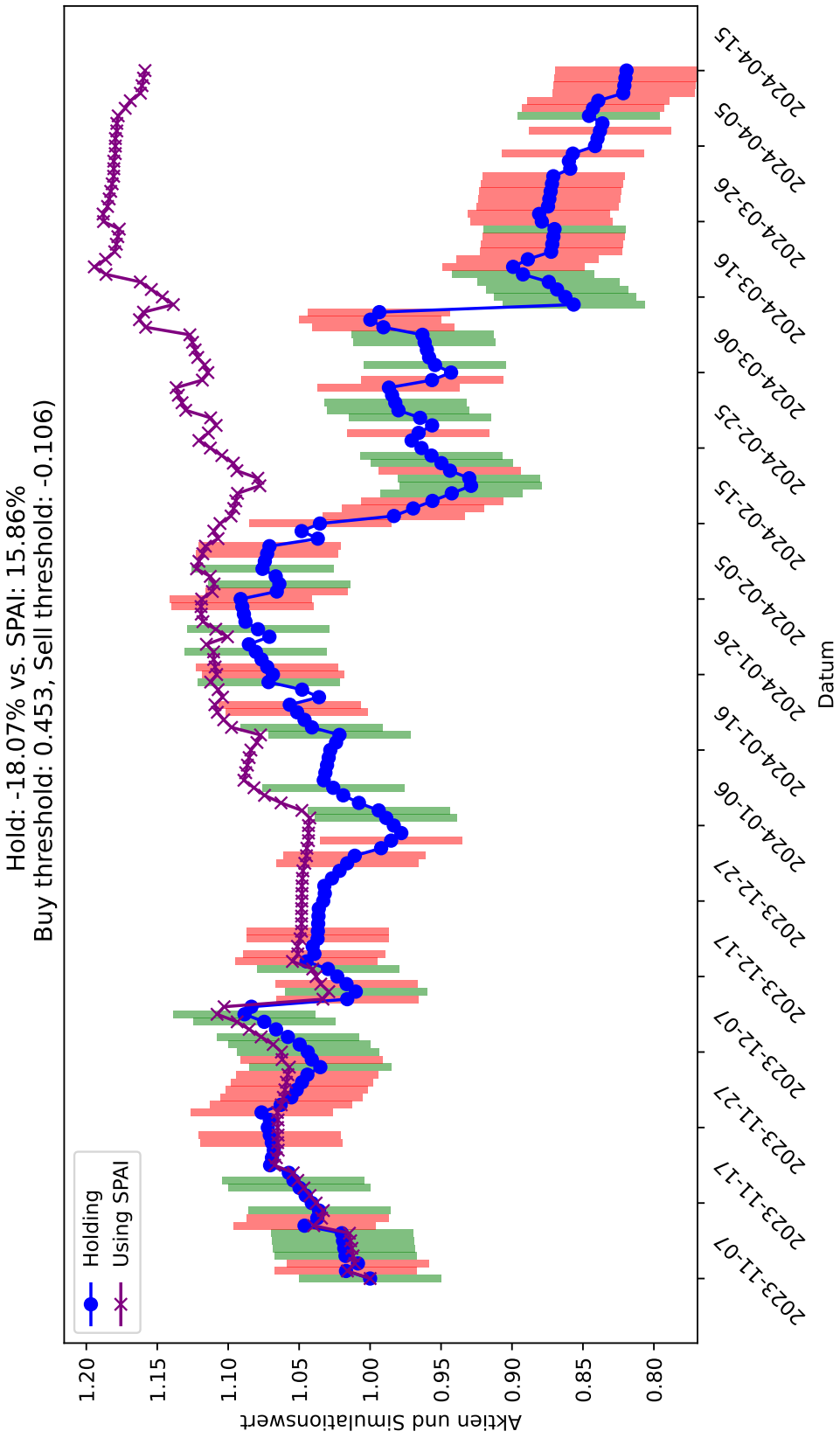

Abbildung 14: Simulation mit Handelssignalen

4 Fazit und Ausblick

Die vorliegende Arbeit demonstriert erfolgreich die Anwendung von Machine Learning-Techniken zur Vorhersage von Aktienpreisveränderungen, speziell am Beispiel von Adobe Inc. Durch die systematische Erfassung und Analyse von Daten, die Implementierung verschiedener Machine Learning Modelle und die Optimierung ihrer Hyperparameter konnte eine effektive Methode zur Generierung von Handelssignalen entwickelt werden. Die Modelle wurden sorgfältig ausgewählt und anhand ihrer Performance mittels Metriken wie dem Mean Squared Error und dem R2 Score evaluiert. Die Anwendung von RandomizedSearchCV zur Optimierung der Hyperparameter und Optuna zur Bestimmung optimaler Schwellwerte für Handelssignale hat sich als praktikabel und effizient erwiesen.

Das beste Modell, das identifiziert wurde, ist die Lineare Regression, die einen Gesamtscore von 4.78 erreicht hat. Die festgelegten Schwellenwerte für Kauf- und Verkaufsentscheidungen sind 0,46 % bzw. -0,011 %. Durch die Anwendung dieser Schwellenwerte konnte ein Depotwert von 1156,76 € erreicht werden, was eine signifikante Steigerung von 33,93 % im Vergleich zur Haltstrategie darstellt.

In Zukunft könnte die Forschung in mehrere Richtungen erweitert werden, um die Genauigkeit und Zuverlässigkeit der Vorhersagen weiter zu verbessern. Erstens wäre die Integration zusätzlicher Datenquellen, wie soziale Medien denkbar, um ein umfassenderes Bild des Marktumfelds zu erhalten und die Modelle auf eine breitere Datenbasis zu stützen. Zweites wäre die Entwicklung einer dynamischen Anpassung der Modellparameter in Echtzeit eine wertvolle Ergänzung, um schnell auf Marktveränderungen reagieren zu können. Letztendlich könnte die Implementierung und das Backtesting der entwickelten Handelssignale auf historischen Daten dazu beitragen, die praktische Anwendbarkeit und Robustheit der Handelsstrategie unter verschiedenen Marktbedingungen zu testen und zu validieren.

Literatur

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*. Andriy Burkov. ISBN: 9781999579517.

Raschka, Sebastian u. a. (2022). *Machine Learning with PyTorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python*. Expert Insight. Birmingham Mumbai: Packt. 741 S. ISBN: 978-1-80181-931-2.