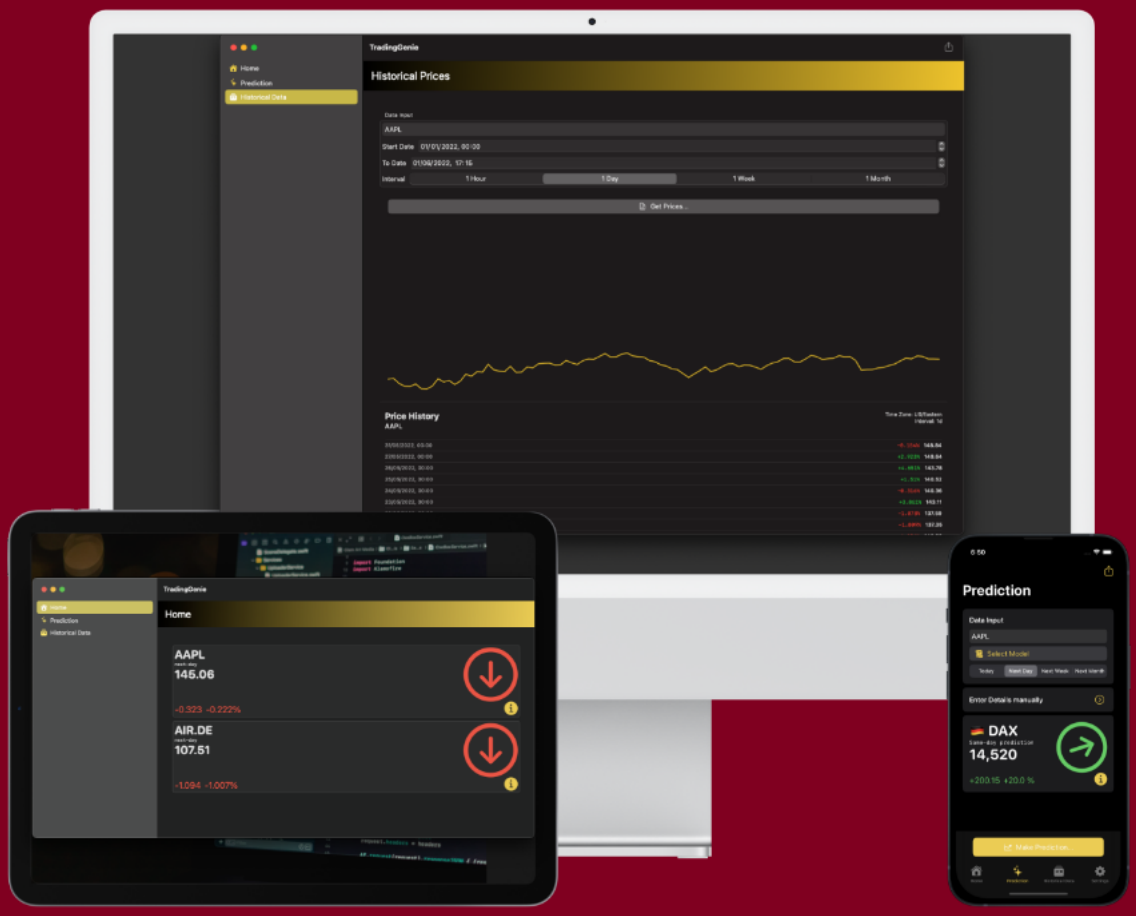
Seminararbeit
Seminarkurs SIA
Fr. Geigis und Hr. Burgbacher
Johannes-Kepler-Gymnasium
Reutlingen



Lassen sich Finanzmärkte mit Machine Learning vorhersagen?

Luca Meurer
luca@mail.excellency.one
Hans-Reyhing-Str. 65/1
72762 Reutlingen
20.06.2022

Inhaltsverzeichnis

1. Einleitung

Digitale Sprachassistenten, die verschiedene Nutzer an ihrer Stimme erkennen können, selbstfahrende Autos, welche in der Lage sind, Menschen und Fahrzeuge zu identifizieren und ihre Bewegungen vorherzusagen, die Erkennung von Hautkrebs mit einer Handy-App: All dies funktioniert mithilfe von Künstlicher Intelligenz (kurz KI). Darunter ist nicht die Vorstellung einer Maschine gemeint, die kognitiv die gleichen Eigenschaften wie das menschliche Gehirn hat, vielmehr ist der logische Prozess der Übertragung von gewonnenen Erkenntnissen auf zukünftige Szenarien gemeint. Diese Technologie hat mich schon immer begeistert.

Die Idee einer „denkenden Maschine" stammt aus dem antiken Griechenland. Aber erst durch die Erfindung elektronischer Computer konnten wichtige Meilensteine in der Entwicklung von KI erreicht werden. Schon 1956 wurde der Begriff „Artificial Intelligence" (kurz AI) durch John McCarthy signifikant geprägt. In diesem Jahr wurde auch die allererste funktionierende KI-Software von Allen Newell, J.C. Shaw und Herbert Simon entwickelt.[1]

Heutzutage nutzen jedoch die meisten Anwendungen Machine Learning (kurz ML), eine Untergruppe der KI. Diese beschäftigt sich mit der Nachahmung der menschlichen Lernprozesse durch Computer, sodass diese aus großen Datenmengen lernen und sich verbessern können, ohne explizit darauf programmiert zu sein.

Schon immer habe ich mich gefragt, ob man diese mächtige Technologie auf Finanzmärkte anwenden könnte. Es sollte möglich sein, relativ zuverlässig Aktienkurse vorherzusagen, was bei korrekter Anwendung zu einem geringeren Risiko und höherem Gewinn bei Investitionen in Aktien führen würde. Aufgrund dessen wird diese Seminararbeit auch ein praktisches Projekt und einen Versuch enthalten. Mein Ziel war es, eine macOS-App zu entwickeln, die mit Hilfe von Machine Learning Finanzmärkte, in diesem Fall Aktienmärkte, vorhersagen kann. Im folgenden soll die Effektivität und Genauigkeit dieser Software überprüft werden, um eine Antwort auf die folgende Hauptfragestellung zu finden:

Wie und mit welcher Genauigkeit lassen sich Preise und Tendenzen auf Finanzmärkten mit Machine-Learning-Algorithmen vorhersagen?

[1] vgl. IBM: Artificial Intelligence.

2. Theoretische Grundlagen

2.1. Künstliche Intelligenz

Künstliche Intelligenz (KI) ist die Übergruppe für alle Technologien, bei denen Maschinen Intelligenz demonstrieren. In der Anwendung der KI wird meistens Informatik mit Datensätzen kombiniert, um das Lösen von Problemen zu ermöglichen. Am Anfang der Forschung in diesem Bereich wurde der Begriff künstliche Intelligenz für das Beschreiben von Maschinen, die menschliches kognitives Verhalten, wie z.B. lernen und Probleme lösen, nachahmen, genutzt.[2] Heute wird KI heute mit Hilfe des Kriteriums Rationalität definiert. Per heutiger Definition trifft die Beschreibung KI auf **alle Maschinen** zu, **die rational handeln** und **Rationalität** in ihren Prozessen **beweisen.**[3] Des weiteren beschäftigt sich die KI besonders in der Forschung mit der Untersuchung und Entwicklung von „intelligent agents“. Dies sind Systeme, die ihre Umgebung wahrnehmen und so handeln, damit die Chancen zum Erreichen des festgelegten Ziels maximiert werden. Es ist möglich, das solche Systeme aus ihrem Verhalten lernen oder wissensbasiert arbeiten.[4] Dabei kann es einfache und komplexe „intelligent agents“ geben, wobei die KI-Forschung sich auf die komplexeren fokussiert.

Verschiedene Untergruppen der KI befassen sich mit der logischen Argumentation und Folgerung (Lösen von logischen Problemen durch Maschinen), Natural Language Processing (Maschinelles Verarbeiten und Verstehen von umfangreichen Texten), Wahrnehmung (Maschinelle Interpretation und Bewertung von Daten in Bezug auf die Umgebung) und dem Lernen (Machine Learning), bei dem die automatische Verbesserung von Algorithmen mit zunehmender Erfahrung (Siehe 2.2) im Vordergrund steht. Machine Learning ist die Technologie, die sehr vielen heutigen KI-Anwendungen zugrunde liegt.

Diese Untergruppen sind der „schwachen" KI bzw. *Artificial Narrow Intelligence* (ANI) zuzuordnen. Dabei wird die KI für spezifische Aufgaben entwickelt. Trotz des Namens ermöglichen diese Technologien robuste Anwendungen, wie z.B. Apple Siri, Amazon Alexa, selbstfahrende Autos von Tesla und die meisten heute existierenden KI-Anwendungen.[2]

---

[2] vgl. IBM: Artificial Intelligence
[3] vgl. Russell et al, S. 2
[4] vgl. Russell et al, S.55

Die „starke“ KI, bestehend aus *Artificial General Intelligence* (AGI) und *Artificial Super Intelligence* (ASI) ist eine theoretische Form der KI, die eine Intelligenz hat, die der menschlichen Intelligenz gleichzusetzen ist (AGI) bzw. diese um ein vielfaches überschreitet (ASI) und ein Bewusstsein mit Selbstkenntnis besitzt. ASI und AGI sind zwar komplett theoretisch, jedoch wird die Entwicklung solcher Systeme schon seit einigen Jahren erforscht.[5]

Beispiele für Anwendungen von KI sind neben Sprachassistenten und selbstfahrenden Autos auch die in den meisten mobilen Endgeräten verbaute Spracherkennung, auch Automatic Speech Recognition (ASR) genannt, Kundenservice-Chatbots und *recommendation engines*, die personalisierte Vorschläge auf z.B. Google ermöglichen.

2.2. Machine Learning (ML)

Machine Learning ist eine große Untergruppe der KI. Dabei geht es allgemein darum, Software zu entwickeln, die aus Daten lernt, ihre Algorithmen kontinuierlich zu verbessern. Das aus den Daten „gelernte“ Modell kann dann Vorhersagen treffen, ohne explizit dazu programmiert zu sein.

Dabei stellt sich die Frage: Wieso muss eine Maschine lernen? Wieso kann sie nicht von Anfang an richtig programmiert werden?

Einerseits können Entwickler nicht alle zukünftigen Szenarien vorhersagen. Zum Beispiel muss ein Roboter, der durch ein Labyrinth navigieren soll, das Layout von jedem Labyrinth lernen, dem er neu begegnet. Andererseits haben Entwickler manchmal auch keine Vorstellung, wie sie selbst eine Lösung programmieren sollen. Bei vielen Problemen, zum Beispiel der Gesichtserkennung, wissen sogar die besten Softwareentwickler nicht, wie diese ohne Machine-Learning-Algorithmen zu lösen sind.[6]

2.3. ML: Funktionsprinzip und Methoden

Die Analyse von Machine-Learning-Algorithmen bzw. der erstellten Modelle und ihrer Leistung lässt sich der *computational learning theory*, einem Bereich der theoretischen Informatik zuordnen. Diese bildet mithilfe von Stochastik und Analysis die mathematische Grundlage für

[5] vgl. Russell et al, S.970-972
[6] vgl. Russel et al, S.1201

Machine Learning. Diese Theorie gibt aufgrund einer limitierten Anzahl an Datensätzen und einer unbekannten Zukunft, also unendlichen Szenarien, keine Garantie für die hohe Leistung von Machine-Learning-Algorithmen her. Darum wird die Leistung und Genauigkeit eines Algorithmus in der Praxis fast immer nachträglich per Versuch gemessen.[7]

In der Praxis wird beim Machine Learning der einer Software zugeführte Datensatz (Beispiel: Spielzeuge, die ein Kind mag - grüner Bereich) genutzt, um mithilfe eines Algorithmus ein Modell des Datenbereichs (Beispiel: Alle Spielzeuge) zu erstellen, indem man Grenzen zieht. Mithilfe dieser Grenzen kann man Vorhersagen treffen (siehe Abb. 1: Das Kind mag mit hoher Wahrscheinlichkeit Spielzeuge im grünen Bereich, also große und komplexe Spielzeuge). Der Datensatz sollte repräsentativ für den Datenbereichs sein, um möglichst zuverlässige Ergebnisse zu bekommen.

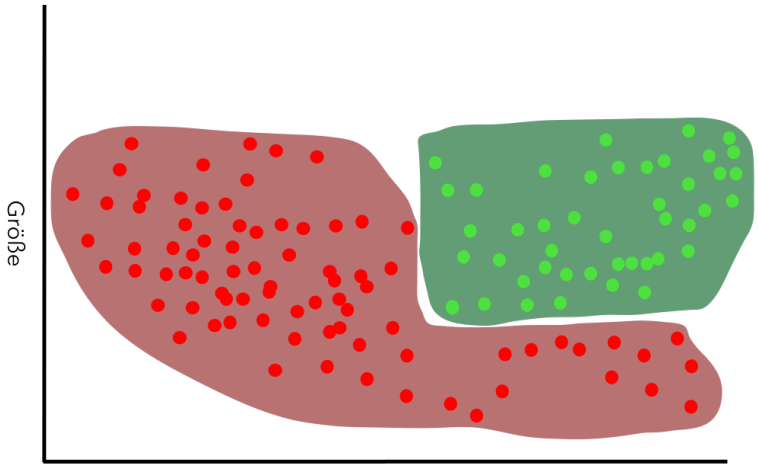


Abb. 1: Schematische Darstellung ML

Machine Learning hat drei grundsätzliche Methoden, die sich je nach Feedback unterscheiden: **Supervised Learning**, **Unsupervised Learning** und **Reinforcement Learning**. Für diese Seminararbeit ist Supervised Learning am relevantesten, da es um die Vorhersage von Werten auf Basis von Beispieldaten geht. Die für diese Seminararbeit nicht relevanten Methoden werden auch kurz erläutert, damit ein genereller Überblick vorhanden ist.[8]

2.3.1. Supervised Learning

Supervised learning (SL) ist in der Anwendung die populärste Methode und wird hauptsächlich für die Vorhersage von numerischen Werten (*regression*) und für Einteilungen (*classification*) genutzt. Wenn der Output ein Wert aus einer endlichen Gruppe an Werten ist, also eine Einteilung in Kategorien stattfindet, spricht man von *classification*. Ein einfaches Beispiel hierfür wäre die Einteilung von Gesichtsfotos in Maskenträger und Nicht-

[7] vgl. Wikipedia: Statistical learning theory
[8] vgl. IBM: What is Machine Learning?

Maskenträger. *Regression* hingegen liefert als Output einen numerischen Wert, beispielsweise die Lufttemperatur des nächsten Tages.[9] Der Begriff *regression* ist in der Forschung der etablierte Begriff, jedoch wären die Begriffe *function approximation* (Funktionsapproximation) oder *numeric prediction* (numerische Vorhersage) passender, da diese die Methodik besser beschreiben.[10]

Beim Supervised Learning enthält der Datensatz (*training data*) eine Reihe von beschrifteten Beispielen (*training examples*). Jedes Beispiel hat ein oder mehrere Inputs (z.B. Fotos) und einen klaren Output, wie z.B. „Auto" oder „Fußgänger". Ein solcher Output wird **Label** genannt. Die Beispiele werden im Algorithmus mathematisch durch Vektoren (*feature vectors*) dargestellt. Der gesamte Datensatz wird durch eine Matrix dargestellt.

Ein Supervised ML-Algorithmus hat 3 Teile:

1) Ein Entscheidungsprozess: Eine Reihe von Berechnungen und anderen Schritten versuchen, eine Schätzung zu einem Muster in den Beispielen zu erstellen. Diese Schätzung ist das vorläufige Modell.

2) Eine Fehlerfunktion: Diese evaluiert die Schätzung durch einen Vergleich mit bekannten Beispielen des Datensatzes und erkennt Fehler. Dadurch kann die Fehlerfunktion die Genauigkeit und Fehlergröße des vorläufigen Modells feststellen.

3) Ein Optimierung- und Aktualisierungsprozess: Der Algorithmus analysiert den Fehler und aktualisiert den Entscheidungsprozess, damit das Modell nächstes Mal genauer ist.

Der Algorithmus durchläuft diese drei Schritte iterativ so lange, bis alle Inputs so gewichtet sind, dass die Zuordnung der Inputs zu einem bestimmten Label eine gewisse Genauigkeit hat.[11] Dadurch hat der Algorithmus eine Funktion „gelernt", mit der man zu neuen Inputs, die nicht Teil der *training data* waren, den passenden Label bestimmen kann. Diese Zuordnungsfunktion wird in einem Modell verpackt, welches man für die Vorhersage von Werten nutzen kann. Dieser Prozess wird in der Fachsprache ***training*** genannt.

[9] vgl. dataiku, S.5-6
[10] vgl. Russell et al. S.1205
[11] vgl. UC Berkeley: What is Machine Learning?

Meistens wird nach dem Training mithilfe eines *Testdatensatzes* die Genauigkeit des Modells überprüft. Dieser besteht meist aus einem abgespaltenen Teil des Trainingsdatensatzes, der vor dem Training entnommen wird und nicht mehr Teil des Trainingsdatensatzes ist.

Der ***error*** und dessen Größe als *Abweichung vom reellen Wert* ist eine wichtige Kenngröße bezüglich der Genauigkeit von Machine-Learning-Algorithmen. Ein Beispiel hierfür wäre die Differenz zwischen dem prognostizierten Aktienwert für ein Zukunftsdatum und dem reellen Aktienwert an diesem Datum. Während der durchschnittliche Error nur auf Trainings- und Testdatensatz bezogen ist, ist der ***mean squared error*** **(MSE)** auf den gesamten Datenbereich bezogen. Er ist der Mittelwert der Quadrate der Errors aller Trainings- bzw. Testbeispiele und im Machine Learning ein Maß der Qualität des Modells. Er ist eine Schätzung des durchschnittlichen Errors des gesamten Datenbereichs, also auch auf Daten ausserhalb des Trainings- und Testdatensatzes bezogen.[12]

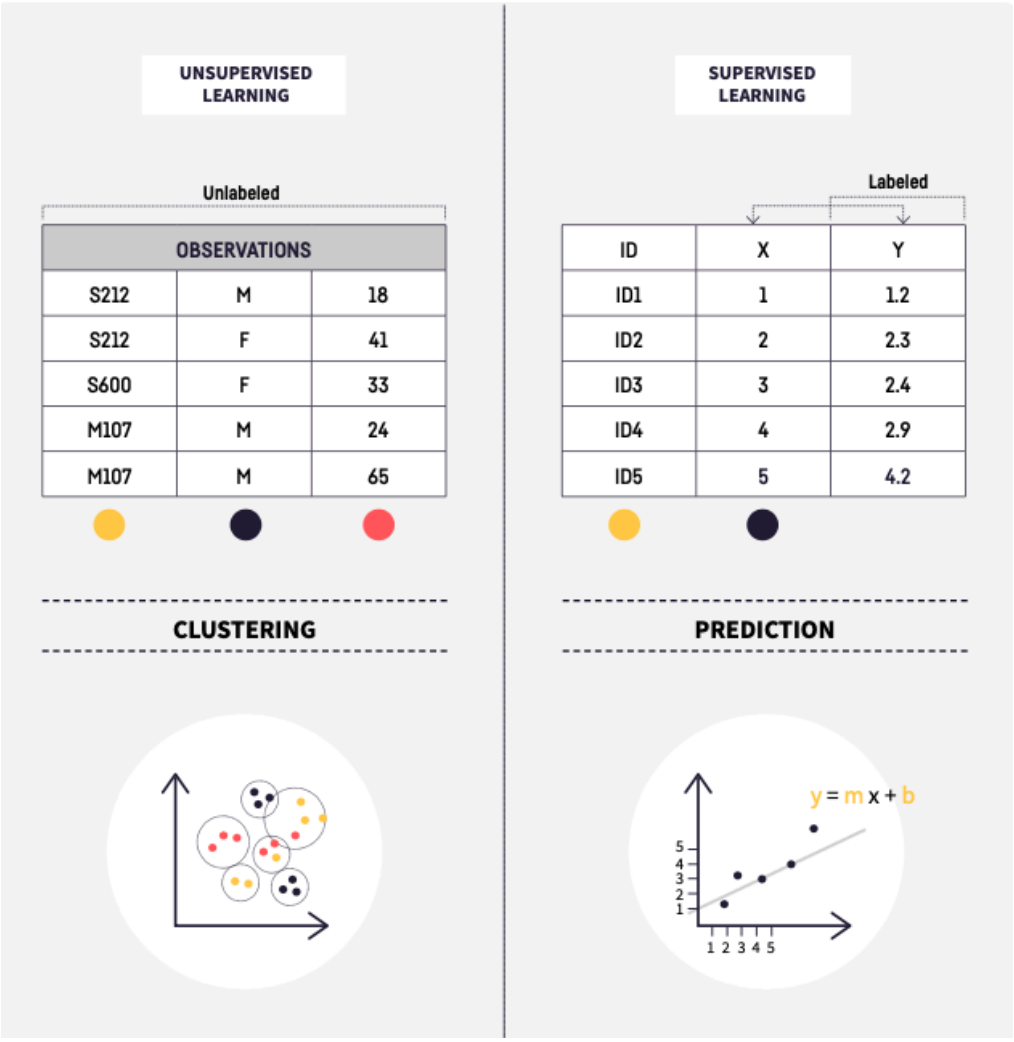


Abb. 2 Supervised und Unsupervised Learning

[12] vgl. IBM: Supervised learning

Supervised learning hat jedoch auch einige Schwierigkeiten:

- Das Training der Modelle kann sehr zeitintensiv sein
- Datensätze können menschliche Fehler enthalten und müssen zuerst zugeordnet (gelabelt) werden

Supervised learning ermöglicht viele moderne Technologien, unter anderem das Einteilen von Emails in Spam und nicht-Spam und das - wie schon erwähnte - Erkennen von Verkehrsteilnehmern in selbstfahrenden Autos.[13]

2.3.2. Unsupervised Learning

Unsupervised Learning (UL) bezieht sich auf die Analyse und Cluster-Einteilung eines Datensatzes (**Clustering**) durch Machine Learning. Damit können unbekannte Muster und Gruppierungen ohne menschlichen Eingriff gefunden werden, die ein mensch normalerweise nicht erkennen würde.

Der Datensatz enthält beim UL nur Inputs und ist weder klassifiziert, kategorisiert noch gelabelt. Es ist kein Feedback vorgesehen, stattdessen werden Zusammenhänge und Muster im Datensatz identifiziert. Anhand dieser Zusammenhänge und Muster kann man Daten in potentiell nützliche Gruppen oder je nach Ähnlichkeit einteilen, um gegebenenfalls unbekannte Zusammenhänge oder Gruppierungen zu erkennen. Auch die Assoziation von Daten, also das finden von Beziehungen und Abhängigkeiten zwischen den Variablen eines Datensatzes ist ein Teil des UL (**Association**).[14] Eine Schwierigkeit des UL ist vor allem die Komplexität der Rechenoperationen, welche längere Trainingszeiten und leistungsfähigere Computer notwendig macht. Es gibt auch eine höheres Risiko, dass nicht akkurate Ergebnisse produziert werden, da die Daten ungeordnet und umbeschriftet verwendet werden und damit auch falsch interpretierbare Daten vorhanden sein könnten.[15]

Anwendungen gibt es beispielsweise im Marketing, indem man Werbung für ein Produkt an eine Gruppe ähnlicher Menschen (z.B. Architekten oder Comedy-Liebhaber) sendet oder diesen Menschen ein ähnliches Produkt anbietet.

---

[13] vgl. IBM: Supervised learning
[14] vgl. Russell et al, S.1206
[15] vgl. IBM: Unsupervised learning

2.3.3. Reinforcement Learning

Reinforcement Learning (RL) ist dem Supervised Learning zwar ähnlich, jedoch werden keine Beispieldaten für das Training genutzt, sondern nur positives und negatives Feedback. Der Algorithmus muss versuchen, selbstständig anhand von vorher definierten Regeln und erlaubten Aktionen ein erfolgreiches zu Ergebnis erzielen. Dabei werden verschiedene Abfolgen so lange ausprobiert, bis das Ergebnis erreicht oder nicht erreicht wird. Der Erfolg sendet ein positives *reward signal* an den Algorithmus, welcher dadurch seine *policy*, also die momentane Verhaltensstrategie ändert. Genauso ist es bei einem Nichterfolg, nur empfängt der Algorithmus ein negatives Signal und ändert dadurch seine Verhaltensweise. Eine Reihe an erfolgreichen Ergebnissen führt zu einer Bestätigung der Verhaltensweise. Der Algorithmus versucht so lange durch Ausprobieren das Ergebnis zu erzielen, bis er eine Verhaltensstrategie gelernt hat, die möglichst immer zum Erfolg führt. Diese Logik der Verhaltensanpassung ist analog zum Schachspielen, wo auch ein Mensch erst nach mehreren Spielen durch Ausprobieren die richtige Methodik kennt. RI findet seine Anwendung vor allem in Spielen, wo Computer gegen Menschen spielen müssen.[16]

2.4. ML: Under- und Overfitting

Je nach Datensatz kann es bei einem ML-Modell zu ***overfitting*** oder ***underfitting*** kommen. Overfitting ist eine Situation, in der ein für die Daten zu komplexes Modell trainiert wurde, da der Algorithmus ungewünschte Elemente des Trainingsdatensatzes in die Modellerstellung einbezieht. Dies führt zu einem übermäßig spezifischen Modell, das nicht auf allgemeine Daten des Datenbereichs außerhalb des Datensatzes anwendbar ist, da es nicht genug verallgemeinert wurde. Es ergibt sich ein geringer *error* im Training, aber ein großer *error* bei der Anwendung des Modells und bei Überprüfung anhand des Testdatensatzes (validation/ evaluation). Hier ist ein hoher *variance error* vorhanden. Gründe hierfür sind unter anderem ein zu langer Trainingsprozess oder ein zu komplexes Modell. Underfitting ist eine Situation, wenn das Modell nicht lang genug trainiert wurde oder wenn der Trainingsdatensatz fehlinterpretiert wird, also keine bzw. falsche Muster und Datenbeziehungen erkannt werden.

[16] vgl. Sutton, R.S. et al, S.2-9

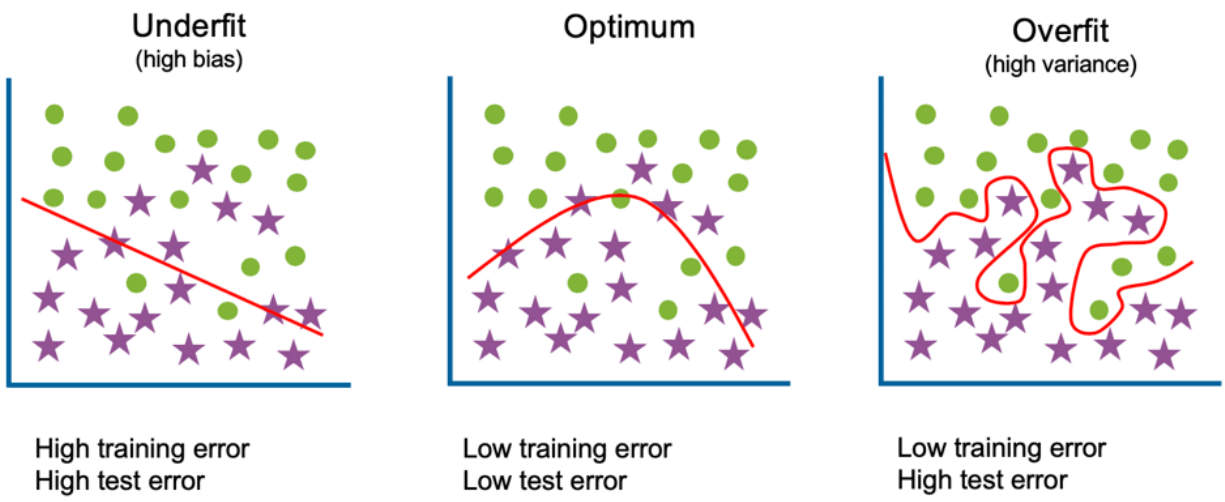


Abb. 3: Underfitting and Overfitting

Es ergibt sich ein hoher *bias error.* Dies ist vor allem bei einem zu kleinen Trainingsdatensatz der Fall. Für ein optimales Machine-Learning-Modell muss also zwischen hoher und niedriger Komplexität und Verallgemeinerung sowie hohem *bias* und hoher *variance* abgewägt werden (**bias-variance tradeoff**), um ein möglichst genaues Modell zu erstellen.[17]

2.5. ML: Verschiedene Algorithmen

Die folgenden Algorithmen finden fast ausschließlich im Supervised Learning Verwendung und werden in der Anwendungsentwicklung am meisten genutzt.

2.5.1. Regression Analysis

Linear regression ist ein Algorithmus der die Beziehung in Form einer Funktion zwischen einer Output-Abhängigkeitsvariable (in Abb. 4 der Immobilienpreis auf dem Mars) und ein oder mehr unabhängigen Input-Variablen (Anzahl Solarpaneelen) identifiziert und die Abhängigkeitsparameter berechnet. Es wird ein linearer Funktionsgraph approximiert, der allen Input-Variablen möglichst genau die Output-Variable zuordnen kann.[18] Bei mehr als einer

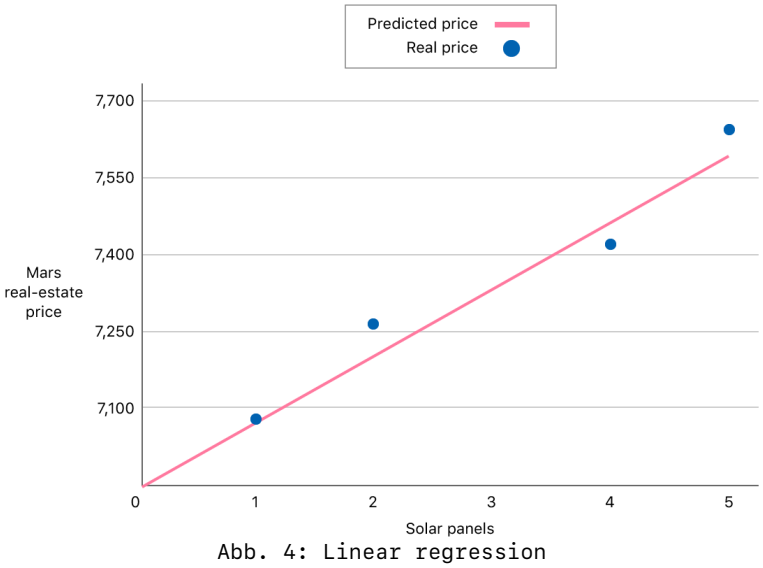


Abb. 4: Linear regression

[17] vgl. IBM: Underfitting
[18] vgl. IBM: Supervised learning

Input-Variable ergibt sich ein Graph in einem mehrdimensionalen Koordinatensystem, z.B. wäre bei 3 Input-Variablen der Graph in einem vierdimensionalen Raum. Die ***polynomial regression*** funktioniert fast gleich wie die linear regression, nur wird kein linearer sondern ein polynomischer Funktionsgraph approximiert.[19]

Die ***logistic regression*** wird zur *classification* verwendet und hat im Gegensatz zur *linear regression* eine binäre Output-Variable, z.B. „wahr“ und „falsch“. Die Input-Variablen werden der Wahrscheinlichkeit, dass eines der beiden Outputs zutrifft, zugeordnet. Dadurch kann eine klare Einteilung anhand eines approximierten Graphen getroffen werden. Das Beispiel in Abb. 5 veranschaulicht dieses Prinzip durch ein Beispiel der Lernzeit als Input und die Wahrscheinlichkeit des Klausurerfolgs als Output. Der Graph kann wie bei der *linear regression* je nach Anzahl der Input-Variablen auch mehrdimensional sein.[20]

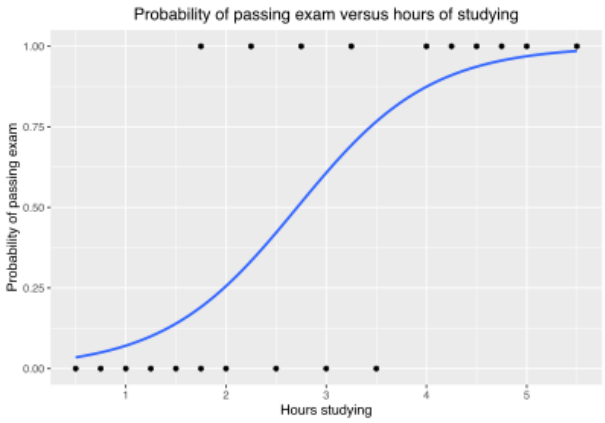


Abb. 5: Logistic regression

2.5.2. Support vector machine

Eine **support vector machine (SVM)** wird meistens zur *classification* genutzt, jedoch ist eine *regression* auch möglich. Eine SVM konstruiert eine Entscheidungsgrenze, an der ein maximaler Abstand zwischen unterschiedlichen Datenpunkten vorhanden ist (*hyperplane*). Die Grenzziehung wird mithilfe sog. *support vectors* (in Abb. 6 gefüllte Symbole) erreicht. Dies sind Datenpunkte, die am nächsten an der Hyperplane dran sind und die Positionierung dieser am meisten beeinflussen. Dieser Prozess hilft der Verallgemeinerung, da die Entscheidungsgrenze die Datenklassen (z.B. Orangen und Äpfel) auf der jeweiligen Seite der

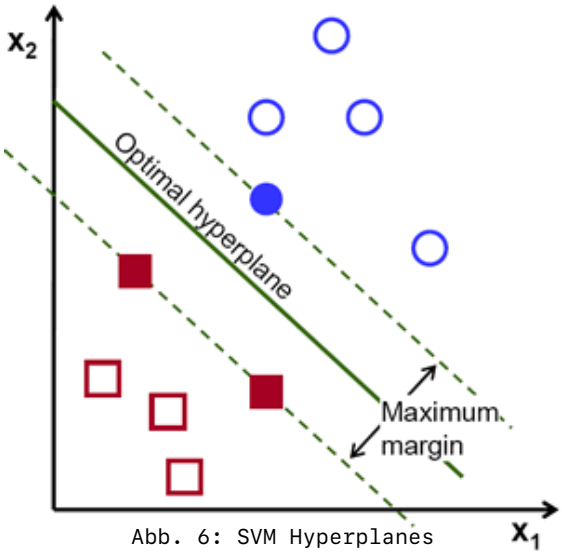


Abb. 6: SVM Hyperplanes

[19] vgl. Wikipedia: Polynomial regression
[20] vgl. Russell et al, S. 1265-1268

Grenze trennt und so eine einfache Einteilung erreichbar ist. Diese Dateneinteilung kann mithilfe des *kernel trick* auch im mehrdimensionalen Raum eingebettet werden (siehe Abb. 7). Dadurch können linear nicht einteilbare Datenmengen durch den mehrdimensionalen Raum eingeteilt werden. Eine SVM gilt als ein *non-probabilistic binary linear classifier*, da ohne Wahrscheinlichkeiten (wie bei *logistic regression*) lineare Grenzen gezogen werden, die Datenpunkte klar in zwei Gruppen einteilen.[21] Eine Vorhersage ist durch die Lokalisierung der Inputdaten in einer bestimmten Gruppe möglich. (z.B. hat die Frucht eine runde Form und orangene Farbe, also findet sich der Datenpunkt in der Orange-Gruppe)

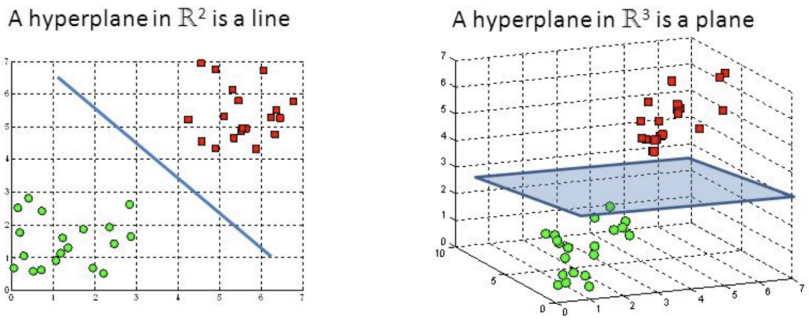


Abb. 7: Mehrdimensionale Hyperplanes

2.5.3. Neural Networks & Deep Learning

Neural networks, auch ***Artificial Neural Networks*** **(ANN)** genannt, sind eine eigene Untergruppe von ML. ANN-Algorithmen verarbeiten die Trainingsdaten durch eine Nachahmung der Interkonnektivität der Neuronen des menschlichen Gehirns. ANN's bestehen aus Schichten von **nodes** bzw. künstlichen Neuronen, die eine Input-Schicht (*input layer*), eine oder mehrere versteckte Schichten (*hidden layers*) und eine Output-Schicht (*output layer*) besitzen. Jede *node* hat eine bestimmte Gewichtung und Schwelle und ist (wie ein Neuron im Gehirn) mit einer weiteren *node* verbunden. Jede einzelne *node* besteht entweder aus einem *linear regressor*, der die Input-Variablen zuordnet, oder einer reinen Übernahme der Inputs. Diese Zuordnung bzw. Inputs werden anschließend gewichtet. Dabei haben die größer gewichteten Variablen eine größere Auswirkung auf den insgesamten Output. Alle

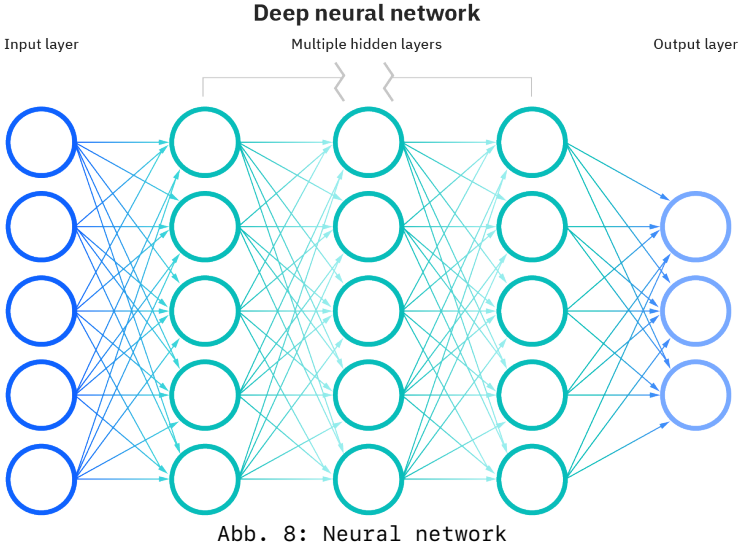


Abb. 8: Neural network

[21] vgl. Gandhi, R: Support Vector Machines

Node-Inputs werden mit den entsprechenden Gewichten multipliziert und dann addiert um den Node-Output zu erhalten. Nur wenn der Output einer *node* über der individuellen Schwelle liegt, wird die *node* aktiviert und sendet Daten an die nächste Schicht (siehe Abb. 8). Dadurch wird der Output der ersten Schicht der Input der zweiten Schicht. Dieser Prozess geschieht in allen *nodes* aller Schichten. Der Output der letzten Schicht ist der Output des Algorithmus. Der Algorithmus gewichtet im Trainingsprozess die Outputs in allen *hidden layers* so, dass die Parameter aller Trainingsbeispiele als Input immer das passende Ergebnis als Output erzeugen. Die Gewichtung der einzelnen Parameter in den *nodes* wird im Modell festgehalten, sodass eine Anwendung auf neue Szenarien einfach möglich ist.[22]

Deep Learning ist ein *neural network*, bei dem mehr als ein *hidden layer* vorhanden ist. Dadurch wird die Komplexität der Entscheidungstruktur erhöht. Dies soll die menschliche Verarbeitung und Klassifizierung von Daten nachahmen, um Objekterkennung in Bildern und Videos effizient und mit hoher Genauigkeit zu ermöglichen.[23]

2.5.4. Decision Trees

Ein **Decision-Tree-Algorithmus** erstellt eine baumartige Entscheidungstruktur, die das Ergebnis vieler Inputs vorhersagen kann. Dabei werden die Regeln der Entscheidungstruktur (siehe Abb. 9) aus den Parametern des Trainingsdatensatzes herausgearbeitet. Im Trainingsprozess werden die Parameter nach ihrer Relevanz sortiert und die Entscheidungstruktur anhand dieser aufgebaut. Der wichtigste Parameter wird zuerst getestet, danach werden die anderen Parameter in abnehmender Relevanz in Unter-Bäumen (*subtree*) getestet. Dann wird diese Entscheidungstruktur mit dem Trainingsdatensatz überprüft. Gegebenenfalls wird eine Anpassung der Relevanz vorgenommen, um die größtmögliche Effizienz zu ermöglichen.[24]

Sleepy	Hungry	Good mood	Productive?
No	No	No	No
No	No	Yes	Yes
No	Yes	No	No
No	Yes	Yes	Yes
Yes	No	No	No
Yes	No	Yes	Yes
Yes	Yes	No	No
Yes	Yes	Yes	No

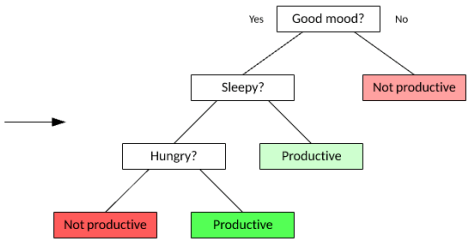


Abb. 9: Decision tree

[22] vgl. IBM: Neural networks
[23] vgl. UC Berkeley: What is machine learning?
[24] vgl. Russell et al, S.1218-1220

In der Anwendung werden die Input-Daten durch die Entscheidungstruktur des erstellten Modells geleitet. Ein Output wird erreicht, indem eine Reihe an Tests von der *root node* („Wurzel" bzw. erster Test) zur *leaf node*, dem Output-Wert, führt (siehe Abb. 9). Dabei entspricht jede *node* einem Test eines Parameters der Input-Daten, dessen Möglichkeiten durch die Abzweigungen gegeben sind. Die leaf nodes am Ende eines *branch* (Zweig) geben den Output an. Decision trees sind aufgrund der einfachen Visualisierbarkeit und Verständlichkeit sehr nützlich, können aber auch je nach Anzahl an Parametern und dessen annehmbaren Werten sehr komplex und groß werden.[25] Es gibt außer dem einzelnen *decision tree* noch weitere Ensemble-Methoden, bei denen mehrere *decision trees* kombiniert werden:

Ein **Boosted-Tree-Algorithmus** kombiniert in Reihe viele *decision trees*. Dabei fokussiert sich jeder *decision tree* darauf, den *error* des Vorgängers durch das Anpassen der eigenen Parameter zu minimieren. Das finale Modell fügt alle *decision trees* zusammen. Diese Methodik wird als ***gradient boosting*** bezeichnet. In der Anwendung bedeutet dies, dass ein Modell vor der Vorhersage versucht den möglichen *error* der Vorhersage zu korrigieren. Dies führt einerseits zu einem extrem genauen und effizienten Modell und damit auch zu einer sehr genauen Vorhersage, jedoch ist der Trainingsprozess sehr zeitintensiv, da *decision trees* nacheinander hinzugefügt und trainiert werden.[26]

Ein **Random-Forest-Algorithmus** nutzt Variationen des Trainingsdatensatzes um mehrere unabhängige decision trees zu trainieren. Die Vorhersagen von allen *decision trees* werden dann zusammengefasst: Bei *classification* wird der Wert genommen, den die Mehrheit der decision trees hat. Eine *regression* nimmt den Mittelwert der Vorhersagen aller *decision trees*. So kann die Genauigkeit der Vorhersagen erhöht und der *variance error* minimiert werden.[27]

[25] vgl. Russell et al, S.1215
[26] vgl. Yildrim, S: Gradient Boosted Decision Trees-Explained
[27] vgl. UC Berkeley: What is machine learning?

2.6. Wirtschaftliche Grundlagen

Ein **Finanzmarkt** beschreibt einen Markt, an dem der Handel mit Wertpapieren stattfindet. Dabei bildet sich der Preis durch Angebot und Nachfrage. Wichtige Beispiele sind Aktienmärkte, Devisenmärkte und Anleihemärkte. Finanzmärkte sind für den reibungslosen Ablauf von Wirtschaftsprozessen in Marktwirtschaften notwendig, da sie Ressourcen zuteilen und für Unternehmen und Gründer Liquidität bereitstellen. Finanzmärkte ermöglichen einen einfachen Handel mit Wertpapieren zwischen Käufern und Verkäufern, da eine hohe Informationstransparenz dafür sorgt, dass die Märkte angemessene und effiziente Preise bilden.[28]

Aktienmärkte sind die allgegenwärtigste Form von Finanzmärkten. Sie ermöglichen Unternehmen, ihre Aktien, also Anteile am Unternehmen, zu notieren. Diese können von Händlern und Investoren gekauft und verkauft werden. Dies geschieht fast immer an einer regulierten öffentlichen Börse. Solche Aktien-Börsen spielen für die Wirtschaft eine wichtige Rolle, da sie die Kapitalgewinnung von Unternehmen durch Aktienauschüttung ermöglichen. Zusätzlich ermöglicht eine Börse einem Investor, einen finanziellen Gewinn durch den Kauf- und Verkauf (zu einem höheren Preis) von Aktien zu erzielen. Aktienbesitzer erhalten in der Regel Dividenden, ein ausgezahlter Anteil am Unternehmensgewinn, welcher einen Kapitalzuwachs für den Aktienbesitzer darstellt.

An einem Aktienmarkt sind auch Börsenmakler beteiligt. Dies sind Dritte, die den Handel von Aktien zwischen Unternehmen und Investoren ermöglichen und vereinfachen. Viele Börsen haben einen Index für die wertvollsten Unternehmen oder eine spezifische Branche (z.B. DAX, Nasdaq 100), der die durchschnittliche Entwicklung des gesamten Marktes angibt.[29]

Die aufgrund von Angebot und Nachfrage im zeitlichen Verlauf fluktuierenden Aktienpreise werden durch einen Aktienkurs beschrieben. An diesem lassen sich zeitliche Entwicklungen erkennen. Viele Methoden zur Analyse, besonders die **technische Analyse**, nutzen die historische Preisentwicklung, das Handelsvolumen (Anzahl gehandelter Aktien innerhalb eines Zeitraums) und die grafische Darstellung eines Aktienkursverlaufs um Schlüsse zur zukünftigen

[28] vgl. Investopedia: Financial Markets
[29] vgl. Investopedia: Stock Markets

Entwicklung des Aktienpreises zu ziehen. Dabei können weitere Indikatoren wie das Kurs-Gewinn-Verhältnis (KGV, Quotient aus Aktienpreis und Unternehmensgewinn) und Moving Averages (MA, Durchschnittspreis eines bestimmten Zeitraums) sehr hilfreich sein.

Die **fundamentale Analyse** hingegen untersucht Geschäftszahlen wie Kapitalfluss, Gewinn, Absatz, Umsatz und Bilanzen sowie die generelle Entwicklung der Wirtschaft und Branche. Anhand dieser Daten kann man entscheiden, ob der Aktienpreis über- oder unterbewertet ist und dann eine Kauf- oder Verkaufsentscheidung treffen.[30]

[30] vgl. Investopedia: When to Use Fundamental, Technical, and Quantitative Analysis

3. Praktische Anwendung - Erstellen einer ML-App

Die praktische Anwendung von Machine Learning in der Softwareentwicklung wird in dieser Seminararbeit anhand einer macOS-App namens **TradingGenie** erläutert. Diese wurde in der Programmiersprache Swift entwickelt und nutzt die *frameworks* (Programmiergerüste) SwiftUI für die Benutzeroberfläche und CoreML/CreateML für die Einbindung von ML. Das Ziel ist, in der App ein ML-Modell zu trainieren, das anhand von Preis- und Handelsvolumens-Daten der letzen fünf Tage eine Vorhersage zum Aktienpreis am Ende des nächsten Tages macht. Zusätzlich soll auch ein Modell, dass anhand dieser Daten keine Preisvorhersage, sondern eine Trendvorhersage (positive/negative Preisentwicklung) macht, trainiert werden. Die Effektivität und Genauigkeit dieser beiden Methoden wird in Kapitel 4 untersucht.

3.1. Programmiersprache Swift

Die Programmiersprache **Swift** ist eine von Apple Inc. und einer Open-Source-Community entwickelte vielseitige, multi-paradigmische kompilierte Programmiersprache. Sie wurde erstmals 2014 als Nachfolger von Objective-C und C veröffentlicht. Swift nutzt den sehr performanten LLVM-Compiler, um auf verschiedenen Geräteplattformen, Chiparchitekturen und Betriebssystemen (iPhone/iPad, Apple Watch, Apple TV, Mac, Linux, arm64, x86) möglichst performant zu sein und wird in C++ entwickelt. Dadurch ist es möglich, innerhalb des Swift-Code auch C, C++ oder Objective-C-Code zu integrieren. So können alle möglichen Bibliotheken aus anderen populären Programmiersprachen in Swift integriert werden, was beim Programmieren die Möglichkeiten weitgehend erweitert.[31]

Einige der *wichtigsten* Merkmale von Swift sind:

Inferred Types: Typen können entweder einfach zugewiesen (`var name: String`) oder aus der Initialisierung erschlossen werden (`var predictionSymbol = „AAPL"` hat den Typ String).

Extensions: Typen können in ihrer Funktionalität durch eine Extension erweitert werden. Diese kann an jedem beliebigen Ort im Code stehen und kann alle möglichen Operationen enthalten:

[31] vgl. Apple Developer: Swift

```
extension Date {
    var dayInYear: Int {
        let cal = Calendar.current
        return cal.ordinality(of: .day, in: .year, for: self) ?? 0
    }
}
var day = Date.dayInYear
```

Native Error Handling: Code, in dem während der Ausführung ein Fehler auftreten kann, muss für diesen Fall eine Ausnahme enthalten (`do {} catch {}`), bzw. entsprechend markiert werden (`try`), um einen Absturz (fatal Error) der Software schon beim Programmieren zu verhindern.

Code Safety: Swift-Code ist extrem sicher und verhindert somit schon beim Kompilieren vorhersehbare Fehler, indem Variablen vor der Nutzung immer initialisiert werden müssen, Arrays und Integers werden automatisch auf Overflow (z.B. zu große Zahl) überprüft und die Arbeitsspeichernutzung wird automatisch gemanagt. Der Syntax ist so angepasst, dass die Intention des Programmierers einfach nachvollziehbar ist. [32]

3.2. SwiftUI: App-Struktur und Benutzeroberfläche

SwiftUI ist ein *framework* von Apple zur deklarativen Erstellung einer App-Struktur mit Benutzeroberfläche (User Interface, kurz UI). Es kann in Verbindung mit UIKit, dem älteren UI-Framework von Apple benutzt werden, da SwiftUI auf UIKit basiert.

Jede SwiftUI-App läuft in einer sog. **App Sandbox**, ein Prinzip, dass auf Kernel-Level das gesamte Betriebssystem schützt, indem eine App nur auf Daten und Ressourcen Zugriff hat, die explizit notwendig sind. Alle Apps auf Apple-Systemen müssen dieses Prinzip einhalten, um eine größtmögliche Sicherheit für die privaten Daten und das Betriebssystem mit allen anderen Apps des Nutzers zu gewährleisten.

Die App-Struktur wird mithilfe von SwiftUI deklarativ beschrieben. Ein erstellter Typ der dem `App` *protocol* entspricht (`TradingGenieApp`) beschreibt die App als Ganzes. Dieser muss mit `@main` markiert werden, damit der Anfangspunkt des Programms gekennzeichnet ist. Eine App kann eine oder `Scene`s enthalten (`WindowGroup`), die verschiedene Aspekte der Benutzeroberfläche entsprechen (z.B. verschiedene Fenster). Datenobjekte (`model`) können mit einem `@StateObject` property wrapper initialisiert werden.

[32] vgl. Swift: About Swift

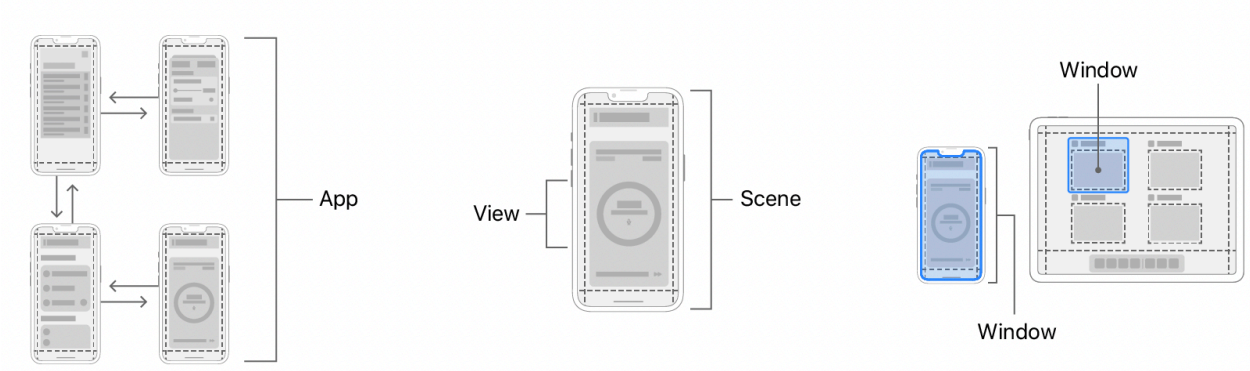


Abb. 10: Aufbau einer SwiftUI-App

Die in Scenes enthaltenen Views (ContentView) können durch einen modifier (.environmentObject(model)) dynamischen Zugriff auf diese Datenobjekte erhalten. Views bilden die Grundlage zum Erstellen der gesamten Benutzeroberfläche, da eine View wiederum andere Views enthalten kann.[33]

```
@main
struct TradingGenieApp: App {
    @StateObject private var model = Model()

    var body: some Scene {
        WindowGroup {
            ContentView().environmentObject(model)
                .preferredColorScheme(.dark)
        }
    }
}
```

Eine View besteht aus einer verpflichtenden body-Variable, die eine oder mehrere andere Views miteinander kombinieren kann. Diese können selbst erstellte Views sein (PlatformNavAdapt) oder in SwiftUI eingebaute Views (ScrollView) sein. Vor dem Body können verschiedene Variablen deklariert und (wenn View-lokal) initialisiert werden. View modifiers können, wie der Name schon sagt, Views z.B. durch mehr Platz auf allen Seiten (.padding()) oder eine andere Schriftart (.font(.system(.headline, design: .rounded))) modifizieren.

```
struct HomeView: View {
    @EnvironmentObject var model: Model

    var body: some View {
        PlatformNavAdapt(title: "Home") {
            ScrollView {
                if !model.predictions.isEmpty {
                    ForEach(model.predictions) { Prediction in
                        PredictionOverview(modelPrediction: Prediction)
                    }

                }
            }.padding()
                .toolbar {} }
    }
}
```

[33] vgl. Apple Developer: Documentation: SwiftUI

Andere wichtige Layout-Elemente sind Listen (`List`), vertikale (`VStack`) und horizontale (`HStack`) Anordnungen, Bilder (`Image()`) und Tasten (`Button`).

3.3. CreateML/CoreML: Integrieren von Machine Learning

Mithilfe des CreateML-*frameworks* wird das Training und Erstellen von ML-Modellen mit Swift für die Anwendungen auf den Apple-Plattformen möglich. Dazu sind zunächst Trainingsdaten notwendig. Diese Trainingsdaten müssen entsprechend organisiert werden, da es sich bei CreateML um *Supervised Learning* handelt. Von diesen Daten muss ein Teil zur Evaluation (*validation*) des trainierten Modells abgespalten werden.

Abb. 11: CreateML workflow

Dann wird ein den Eingangsdaten (Bilder, Tabellendaten, Bewegungen, Sound) entsprechendes Modell (Classifier/Regressor) trainiert. Um den optimalen Algorithmus zu finden, werden alle verschiedenen Trainingsalgorithmen ausprobiert, es wird jeder Trainingsalgorithmus durchlaufen. Für einen Tabular Regressor (Regressor für Tabellendaten) sind dies: Linear Regressor und Boosted Tree.[34] Ein Tabular Classifier (Einordnung von Tabellendaten) kann die Algorithmen Decision Tree, Random Forest, Boosted Tree, Logistic regression und SVM nutzen.[35] Die App nutzt für die **Preisvorhersage** einen **Tabular Regressor**, da es sich um eine numerische Vorhersage handelt. Für die **Trendvorhersage** verwendet die App einen **Tabular Classifier**, da eine Einteilung in einen positiven oder negativen Trend stattfindet.

Nach dem Trainingsprozess für alle Algorithmen werden diese anhand der vorher abgespaltenen Evaluationsdaten auf die Genauigkeit untersucht. Bei einer *classification* wird hier der Anteil an falsch eingeordneten Datenpaaren ausgegeben, bei einer *regression* der *mean square error* und der maximale Error (Definition siehe Kap. 2.3.1). Anschließend wird im Ausgabefenster eine Auflistung der Trainingsgenauigkeit aller Algorithmen dargestellt. CreateML wählt automatisch den Algorithmus mit dem genausten Modell aus. Diese Funktion

[34] vgl. Apple Developer: Documentation: MLRegressor
[35] vgl. Apple Developer: Documentation: MLClassifier

ist allerdings nur auf macOS verfügbar, da die (meistens) geringere Rechnerleistung auf iOS/iPadOS-Geräten diese Funktion nicht ermöglicht.

In der konkreten Anwendung sieht dieser Workflow wie folgt aus:

Organisation der von einem Web-Server heruntergeladenen und von JSON (JavaScript Object Notation) in Swift konvertierten Trainingsdaten in einer `MLDataTable`, die ein Array an Datenspalten (`namedColumns`) enthält:

```
let data = try! MLDataTable(namedColumns: […])
```

Aufspaltung der Daten in Trainings- und Evaluationsdaten:

```
let (trainingData, testingData) = data.randomSplit(by: 0.8)
```

Training eines regressors (`MLRegressor`) und classifiers (`MLClassifier`) , es werden alle Algorithmen ausprobiert und evaluiert (`.evaluation(on: testingData)`):

```
let pricePredictor = try? MLRegressor(trainingData: trainingData, targetColumn: „actualClose“)
let priceEvaluation = pricePredictor?.evaluation(on: testingData)
let trendPredictor = try? MLClassifier(trainingData: trainingData, targetColumn: „trendUp“)
let trendEvaluation = trendPredictor?.evaluation(on: testingData)
```

Um Rechnerleistung zu sparen kann in einer *shipping application* (veröffentlichte App) der effektivste Algorithmus (z.B. `MLBoostedTreeClassifier`) mit mehreren Iterationen (beschrieben durch `MLLinearRegressor.ModelParameters`) verwendet werden. So muss nicht jeder Algorithmus in der User-Anwendung neu ausprobiert werden, was das Benutzererlebnis aufgrund von hängender, nicht reagierender Software (in der Fachsprache *hang* genannt) extrem verschlechtern kann. Es wird also direkt der genaueste Algorithmus zum Training genutzt. Zum Beispiel:

```
let priceParams = MLLinearRegressor.ModelParameters(maxIterations: 2000)
let pricePredictor = try? MLLinearRegressor(trainingData: trainingData, targetColumn:
"actualClose", parameters: priceParams)

let trendParams = MLBoostedTreeClassifier.ModelParameters(maxIterations: 2000)
let trendPredictor = try? MLBoostedTreeClassifier(trainingData: trainingData, targetColumn:
„trendUp", parameters: trendParams)

let priceEvaluation = pricePredictor?.evaluation(on: testingData)
let trendEvaluation = trendPredictor?.evaluation(on: testingData)
```

Eine Vorhersage mithilfe des neu trainierten und evaluierten Modells kann durch den `.predictions(from: )` -modifier, der in CoreML enthalten ist, erreicht werden. Die Resultate der Vorhersage werden einem Array (`pricePrediction?.doubles?/`

`trendPrediction?.ints?`) entnommen und (in der TradingGenie-App) auf Variablen des dynamischen Datenobjektes geschrieben:

```
let pricePrediction = try! pricePredictor?.predictions(from: predictionData)
let trendPrediction = try! trendPredictor?.predictions(from: predictionData)

predictedClose = pricePrediction?.doubles?.element(at: 0) ?? 0
if (trendPrediction?.ints?.element(at: 0) == 1) {
   predictedTrendUp = true
} else {
   predictedTrendUp = false
}
```

Diese beschriebenen Variablen könne in einer View angezeigt werden, um dem Nutzer die relevanten Informationen der Vorhersage zu übermitteln.

3.4. Persönliche Erfahrungen bei der Entwicklung der App

Im Nachhinein war das Projekt für mich sehr herausfordernd. Obwohl ich schon Erfahrungen mit anderen Projekten bezüglich Benutzeroberfläche hatte, war insbesondere die Visualisierung der Vorhersagen mit dem Ziel, einen möglichst großen Nutzen für den User zu erreichen, ein großer Entwicklungsprozess. Da dies meine erste App mit Machine Learning war, stellte das Lernen der CreateML- und CoreML-frameworks den anspruchsvollsten aber auch bereicherndsten Teil des gesamten Prozesses dar. Das mir angeeignete Wissen lässt sich nun ideal für neue App-Ideen nutzen.

Im Prozess der App-Entwicklung habe ich zunächst eine einfache Ansicht mit Eingaben für die wichtigsten Werte und eine Ansicht für die Resultate der Vorhersage ohne Datenverarbeitung oder -speicherung programmiert. Anschießend habe ich die App mit einer Home-View und einer View für den historischen Preisverlauf einer Aktie erweitert. Um die Navigation zwischen diesen Views zu ermöglichen habe ich ein Tab-Menü integriert. Einige Wochen später habe ich das Datenobjekt im Hintergrund mit allen Variablen und dem dynamischen Zugriff entwickelt. Dieses Datenobjekt musste im Verlauf der App-Entwicklung viele Versionen durchlaufen, v.a. wegen Datenerweiterung sowie Funktionserweiterungen. Als nächstes habe ich das Herunterladen von Aktiendaten programmiert. Hierbei gab es zweimal hintereinander Probleme mit dem Web-Server, der nicht alle Aktiendaten zur Verfügung hatte. Dies bedeutete auch ein Überarbeiten des Konvertierungsprozesses von JSON- zu Swift-Objekten, was ziemlich ärgerlich war. Dennoch habe ich schließlich eine Lösung gefunden und konnte das

Herunterladen und Konvertieren von den Aktiendaten anhand der Anzeige von historischen Aktienpreisen in der App testen. Nachdem dies zufriedenstellend funktionierte, habe ich mich an das Integrieren von ML gemacht. Dabei habe ich mich von Anfang an für ein Supervised Learning-Modell entschieden, da ich anhand von gekennzeichneten Daten eine Vorhersage treffen wollte. Ich musste allerdings in letzter Minute noch die Plattform ändern, da ich herausgefunden hatte, dass CreateML auf manchen iOS-Geräten (inklusive meinem) nur eingeschränkt läuft, weil viele iOS-Geräte nicht die Rechnerleistung für ML-Algorithmen haben. Da SwiftUI auf allen Apple-Plattformen läuft, konnte ich noch kurzfristig die Plattform der App auf macOS ändern, um die Integration von CreateML zu ermöglichen. Dabei war das Programmieren der Funktionen zum Training und zur Vorhersage relativ einfach, jedoch musste ich zuerst alle Klassen und Methoden verstehen und kennenlernen. Dabei ist mir oft erst beim Kompilieren des Programms aufgefallen, dass sich Fehler eingeschlichen haben. Besonders das Konvertieren der Trainingsdaten zum Tabellendatenformat war ein aufwändiges hin-und-her, das mehrere Tage in Anspruch nahm. Das Entnehmen der Vorhersagedaten war auch nur durch Ausprobieren möglich, da die Dokumentation hier sehr kryptisch formuliert war. Der Trainingsprozess mithilfe aller Algorithmen und die darauffolgende Implementation des genausten Algorithmus war relativ einfach machbar, da das framework eine tabellenartige Übersicht aller Algorithmen und dessen Genauigkeit (siehe Anhang II) liefert.

Für die Zukunft habe ich gelernt, dass die Fehlerbehebung doch länger dauert als oft erwartet und dass ich dafür die Hälfte der Zeit einplanen sollte. Dennoch bin ich sehr stolz über die entwickelte App und dass diese überhaupt funktioniert. Der gesamte Code für die App ist im Anhang I einsehbar.

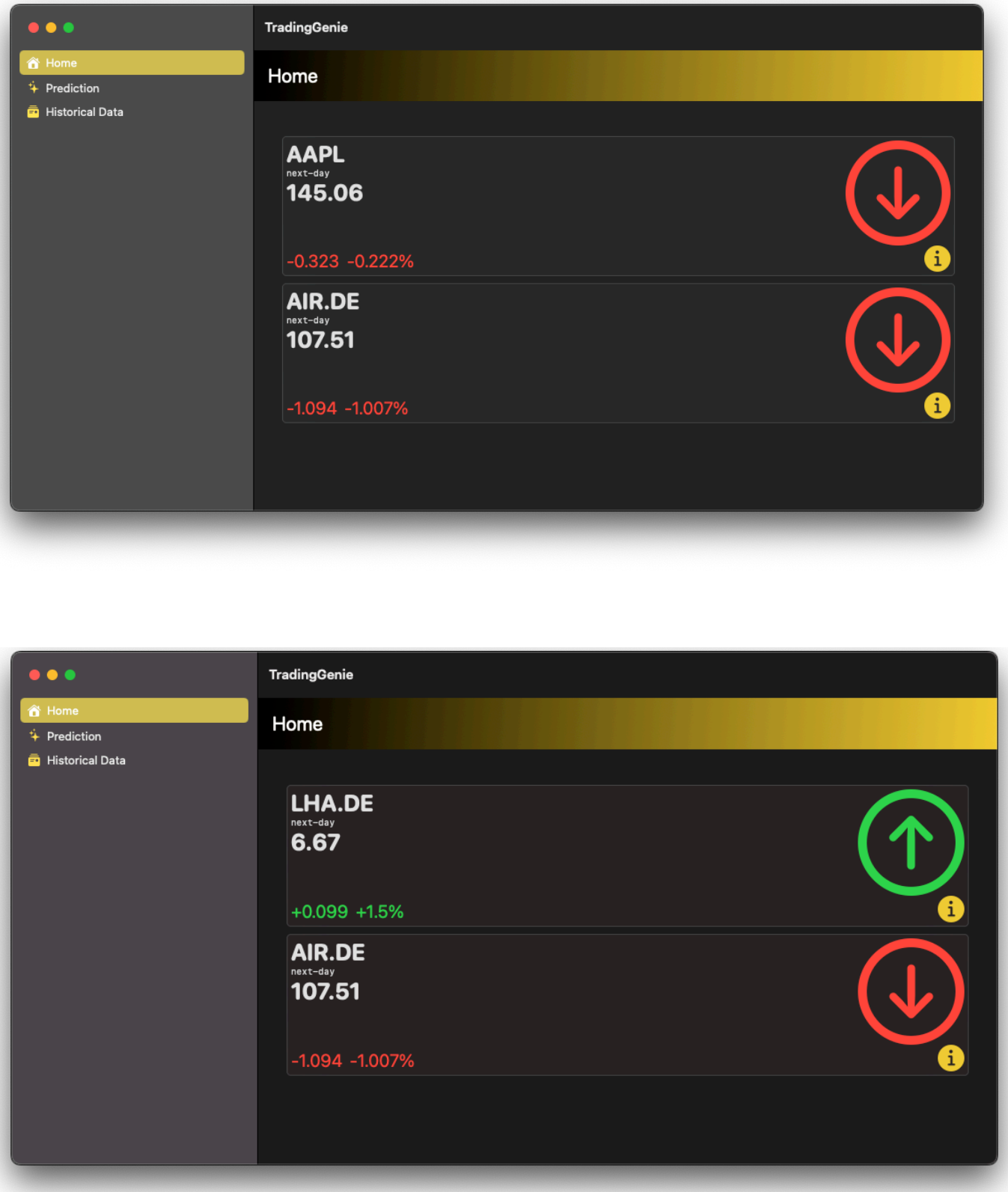


Abb. 12: Screenshots der fertigen App

4. Genauigkeitsermittlung der entwickelten ML-App

4.1. Ziel

In Anlehnung an die Hauptfragestellung der Seminararbeit, wie sie in der Einleitung beschrieben wurde, soll dieser Versuch die Genauigkeit einer ML-Vorhersage bezüglich Preis und Trend (positives/negatives Preiswachstum) von zwei Einzelaktien am Ende des nächsten Handelstages ermitteln und mit welchem Algorithmus dies erreicht wurde. Mithilfe des Ergebnisses soll eine Übertragung der gewonnenen Erkenntnisse im Allgemeinen diskutiert werden, um eine Aussage über die Vorhersagbarkeit von Finanzmärkten mit Machine Learning zu treffen.

4.2. Durchführung

Der Versuch soll mithilfe der in Kapitel 3 beschriebenen App durchgeführt werden. Die Trainings- und Evaluationsprozesse von CreateML geben durch die Evaluation eine Zuordnungsgenauigkeit her. Diese ist bei der *regression* durch den *mean square error* und den *maximum error* gegeben. Bei der *classification* wird durch die *classification accuracy* der Anteil der richtig zugeordneten Trainingspaaren am Evaluations-Datensatz angegeben. Mithilfe der App soll zuerst für die *Tabular Regression* des Aktienpreises und die *Tabular Classification* des Trends anhand des Apple-Aktienpreises der genauste Algorithmus ermittelt werden. Dieser wird für den Trainingsprozess des Vorhersagemodells der Apple- und Airbus-Aktien verwendet. Anschießend wird durch die Evaluierung dieser Modelle die durchschnittliche Genauigkeit der Trend- und Preisvorhersage ermittelt. Der Datensatz besteht aus 200 Beispielen. Ein Beispiel enthält den richtigen Preis und den richtigen Trend für ein Datum und die Anzahl des Tages im Jahr. Darüber hinaus sind auch das Handelsvolumen, Höchst- und Tiefstpreis und der Preis am Anfang und Ende des Handelstages für die vorherigen fünf Tage Teil eines Beispiels. Insgesamt gibt es also pro Beispiel 26 Datenpunkte.

4.3. Beobachtung

Bei der Ermittlung des genauesten Algorithmus für die **Preisvorhersage** war die *Linear regression* mit einem MSE von 4,7 und einem maximalen Error von 10,11 in der Evaluation (Validation) für die Preisvorhersage am genauesten. Dabei beträgt die Abweichung vom

Durchschnittspreis ±3%. Der Boosted Tree-Algorithmus ist auch bei mehreren Iterationen ungenauer (siehe Abb. 13 und Anhang II: Training zur Ermittlung des besten Algorithmus).

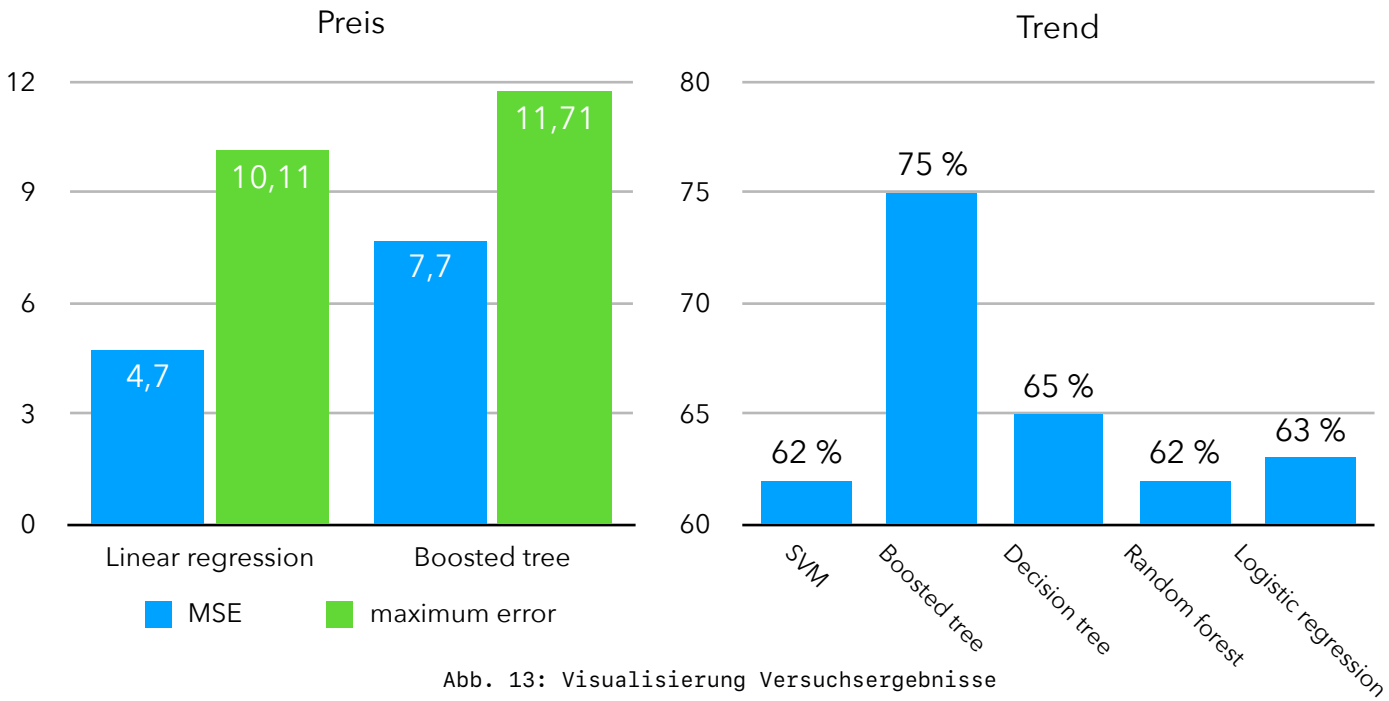


Abb. 13: Visualisierung Versuchsergebnisse

Bei der **Trendvorhersage** war der Boosted Tree-Algorithmus mit einer *Validation Accuracy* von 75% bei mehreren Iterationen in der Evaluation am genauesten. Ausserdem lässt sich feststellen, dass die Genauigkeit mit zunehmenden Iterationen weitestgehend gleich bleibt. (siehe Abb. 13 und Anhang II: Training zur Ermittlung des besten Algorithmus).

Die Anwendung dieser ermittelten Algorithmen auf die Aktien von Apple Inc und Airbus SE liefert folgende Ergebnisse (Original siehe Anhang II: Training für AAPL/AIR.DE):

Für die Apple-Aktie hat die Trendvorhersage bei einer großen Menge an Iterationen (2000) eine Genauigkeit von 80%. Die Preisvorhersage hat einen MSE von 3,9 und einen maximalen Error von 7,05. Die durchschnittliche Abweichung beträgt ±2,3%.

Die Trendvorhersage der Airbus-Aktie hat bei der gleichen Anzahl an Iterationen eine Genauigkeit von 70%. Die Preisvorhersage hat einen MSE von 2 und einen maximum error von 4. Die durchschnittliche Abweichung ist hier ±1,9%. Allgemein ist festzustellen, dass ab einer gewissen Anzahl an Iterationen sich die Genauigkeit bei der Trendvorhersage nicht erhöht.

4.4. Ergebnis

Die Ergebnisse bezüglich der Genauigkeit verschiedener *classification*- und *regression*-Algorithmen decken sich mit Untersuchungen der Stanford University zu diesem Gebiet[36]. Die fehlerkorrigierende Art des Boosted-Tree-Algorithmus ist bei der Vorhersage des Preistrends eines Aktienpreises ideal. Vorhersagefehler durch die zufällige Entwicklung von Aktienpreisen werden durch die Reihe an Decision Trees korrigiert.

Die Genauigkeit von >70% in der Trendvorhersage der beiden Aktien durch einen Boosted Tree-Algorithmus deckt sich mit anderen wissenschaftlichen Untersuchungen und kann somit als allgemein geltend erachtet werden.[37]

Es ist nachvollziehbar, dass sich der Preis am besten durch einen Linear Regressor vorhersagen lässt, da sich die Preisentwicklung eines Aktienpreises über wenige Tage mit einer linearen Funktion am besten approximieren lässt. Der Linear Regressor hat sich durch den Versuch als genaueste Methode zur Preisvorhersage erwiesen, jedoch trifft die Vorhersage mit einem MSE von >1,9% meistens nur ungefähr zu. Eine exakte Vorhersage des Preises ist fast unmöglich, da es unendlich Möglichkeiten geben kann.

Da Finanzmärkte viele ähnliche Eigenschaften haben, kann man die im Versuch gewonnenen Erkenntnisse zu Aktienmärkten auf Finanzmärkte im Allgemeinen übertragen. Die präzise Genauigkeit ist aber abhängig von der unterschiedlichen Datenlage und den geringen, aber dennoch existierenden Unterschieden, insbesondere beim Verhalten der Preisentwicklung in Aktienmärkten.[37]

In einem Investment-Kontext ist die Trendvorhersage als Entscheidungshilfe bei Investmententscheidungen hilfreicher, da eine informiertere und weniger risikobehaftete Entscheidung (kaufen/verkaufen) getroffen werden kann. Die Preisvorhersage kann genutzt werden, um die Trendvorhersage zu bestätigen.

Der Versuch kann als erfolgreich erachtet werden, da in mehr als 50% der Fälle eine erfolgreiche Trendvorhersage gemacht wurde und die Preisvorhersage nur eine Abweichung vom realen Wert im unteren einstelligen Prozentbereich hat.

---

[36] siehe Zhang, Y. et al, S. 2 und Zhang, T. et al, S. 4
[37] vgl. Zhang, T. et al, S. 3

5. Schlussfolgerung und Ergebnis

In dieser Seminararbeit wurden die Grundlagen von Machine Learning erklärt und anhand eines Beispiels angewendet. Die Anwendung von ML ist für jeden eingelesenen Softwareentwickler machbar. Hauptfrage dieser Seminararbeit war, ob man Finanzmärkte mit ML vorhersagen kann. Folgende Erkenntnisse aus dieser Seminararbeit beantworten diese Frage:

I. Die Vorhersage von Finanzmärkten ist mit Machine Learning, genauer Supervised Learning, grundsätzlich möglich.

II. Die Vorhersage der Preisentwicklung bzw. dem Trend von Aktienmärkten ist mit hoher (>70%) Genauigkeit möglich und ist zuverlässiger als die Preisvorhersage, welche als gute Bestätigung dienen kann.

III. Die Trendvorhersage ist mit einem Boosted-Tree-Algorithmus am besten möglich, da dieser die höchste Genauigkeit liefert.

IV. Die Preisvorhersage hat bei der Benutzung eines Linear-Regressor-Algorithmus die geringste Abweichung vom realen Wert.

V. Die sehr genaue Vorhersage eines Preises ist fast unmöglich. Preisvorhersagen treffen meistens nur ungefähr zu.

Ich persönlich finde dass Machine Learning eine große Chance - nicht nur in der Finanzbranche - darstellt. Diese Seminararbeit hat gezeigt, wie die Anwendung von Machine Learning in der Aktienmarktvorhersage einfach und effizient möglich ist. Dennoch gilt es zu klären, welche Auswirkungen dies bei großflächiger Anwendung auf Finanzmärkte und die Gesellschaft haben wird und ob mit größerer Rechnerleistung durch Neural Networks oder Unsupervised Learning in diesem Bereich neue Erkenntnisse gewonnen werden können. Ich sehe eine Möglichkeit für weitere Forschungsprojekte bezüglich Machine Learning und Finanzmärkten.

6. Quellenverzeichnis

6.1. Bildquellen

- Abb. 1: Schematische Darstellung ML. Luca Meurer: Eigene Grafik
- Abb. 2: Supervised und Unsupervised Learning. Aus: dataiku: Machine Learning Basics: An Illustrated Guide
  for Non-Technical Readers. https://content.dataiku.com/ml-basics (letzter Zugriff 28.05.2022)
- Abb. 3: Underfitting and Overfitting. Aus: IBM: What is Underfitting? https://www.ibm.com/cloud/learn/underfitting (letzter Zugriff: 28.05.2022)
- Abb. 4: Linear regressor. Aus: Apple Developer: Documentation: MLRegressor. https://developer.apple.com/documentation/createml/mlregressor (letzer Zugriff 29.05.2022)
- Abb. 5: Logistic regression. Aus: Wikipedia: Logistic regression. https://en.wikipedia.org/wiki/Logistic_Regression (letzer Zugriff 27.05.2022)
- Abb. 6: SVM Hyperplanes. Aus: Gandhi, R (2018): Support Vector Machine. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (letzter Zugriff 28.05.2022)
- Abb. 7: Mehrdimensionale Hyperplanes. Aus: Gandhi, R (2018): Support Vector Machine. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (letzter Zugriff 28.05.2022)
- Abb. 8: Neural networks. Aus: IBM (2020): Neural Networks. https://www.ibm.com/cloud/learn/neural-networks (letzter Zugriff 28.05.2022)
- Abb. 9: Decision tree. Aus: IBM Developer (2017): Models for machine learning, https://developer.ibm.com/articles/cc-models-machine-learning/#reinforcement-learning (letzter Zugriff 28.05.2022)
- Abb. 10: Aufbau einer SwiftUI-App. Aus: Apple Developer (o.J.): Tutorials: Develop Apps for iOS: Responding To Events.https://developer.apple.com/tutorials/app-dev-training/responding-to-events (letzer Zugriff 30.05.2022)
- Abb. 11: CreateML workflow. Aus: Apple Developer (o.J.): Documentation: Create ML. https://developer.apple.com/documentation/createml (letzer Zugriff 02.06.2022)
- Abb. 12: Screenshots der fertigen App. Luca Meurer: Eigenes Bild
- Abb. 13: Visualisierung Versuchsergebnisse. Luca Meurer: Eigene Grafik

6.2. Literaturquellen

- Apple Developer (o.J.): Documentation: Core ML. https://developer.apple.com/documentation/coreml (letzer Zugriff 29.05.2022)
- Apple Developer (o.J.): Documentation: Create ML. https://developer.apple.com/documentation/createml (letzer Zugriff 02.06.2022)

- Apple Developer (o.J.): Documentation: Creating a Model from Tabular Data. https://developer.apple.com/documentation/createml/creating_a_model_from_tabular_data (letzer Zugriff 02.06.2022)
- Apple Developer (o.J.): Documentation: MLBoostedTreeClassifier. https://developer.apple.com/documentation/createml/mlboostedtreeclassifier (letzer Zugriff 03.06.2022)
- Apple Developer (o.J.): Documentation: MLClassifier. https://developer.apple.com/documentation/createml/mlclassifier (letzer Zugriff 03.06.2022)
- Apple Developer (o.J.): Documentation: MLDataTable. https://developer.apple.com/documentation/createml/mldatatable (letzer Zugriff 03.06.2022)
- Apple Developer (o.J.): Documentation: MLLinearRegressor. https://developer.apple.com/documentation/createml/mllinearregressor (letzer Zugriff 03.06.2022)
- Apple Developer (o.J.): Documentation: MLRegressor. https://developer.apple.com/documentation/createml/mlregressor (letzer Zugriff 29.05.2022)
- Apple Developer (o.J.): Documentation: SwiftUI. https://developer.apple.com/documentation/SwiftUI (letzer Zugriff 29.05.2022)
- Apple Developer (o.J.): Swift. https://developer.apple.com/swift/ (letzer Zugriff 29.05.2022)
- Apple Developer (o.J.): SwiftUI. https://developer.apple.com/xcode/swiftui/ (letzer Zugriff 29.05.2022)
- Apple Developer (o.J.): Tutorials: Develop Apps for iOS. https://developer.apple.com/tutorials/app-dev-training (letzer Zugriff 30.05.2022)
- Apple Developer (o.J.): Tutorials: Introducing SwiftUI. https://developer.apple.com/tutorials/swiftui (letzer Zugriff 01.06.2022)
- dataiku (2021): Machine Learning Basics: An Illustrated Guide for Non-Technical Readers. https://content.dataiku.com/ml-basics (letzter Zugriff 28.05.2022)
- Gandhi, R (2018): Support Vector Machine. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (letzter Zugriff 28.05.2022)
- Hudson, P. (2018): How to perform regression analysis using Create ML. https://www.hackingwithswift.com/articles/145/how-to-perform-regression-analysis-using-create-ml (letzter Zugriff 01.06.2022)
- IBM (2020): Artificial Intelligence (AI). https://www.ibm.com/cloud/learn/what-is-artificial-intelligence (letzer Zugriff 01.05.2022)
- IBM (2020): Neural Networks. https://www.ibm.com/cloud/learn/neural-networks (letzter Zugriff 28.05.2022)
- IBM (2020): Supervised learning. https://www.ibm.com/cloud/learn/supervised-learning (letzter Zugriff 28.05.2022)

- IBM (2020): Unsupervised learning.https://www.ibm.com/cloud/learn/unsupervised-learning (letzter Zugriff 28.05.2022)
- IBM (2020): What is machine learning?. https://www.ibm.com/cloud/learn/machine-learning (letzer Zugriff 01.05.2022)
- IBM (2021): What is overfitting?. https://www.ibm.com/cloud/learn/overfitting (letzter Zugriff 28.05.2022)
- IBM (2021): What is underfitting?. https://www.ibm.com/cloud/learn/underfitting (letzter Zugriff 28.05.2022)
- IBM Developer (2017): Models for machine learning, https://developer.ibm.com/articles/cc-models-machine-learning/#reinforcement-learning (letzter Zugriff 28.05.2022)
- Investopedia (2021): Basics of Algorithmic Trading: Concepts and Examples. https://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp (letzter Zugriff 28.05.2022)
- Investopedia (2021): Fundamental Analysis. https://www.investopedia.com/terms/f/fundamentalanalysis.asp (letzter Zugriff 28.05.2022)
- Investopedia (2021): When to Use Fundamental, Technical, and Quantitative Analysis. https://www.investopedia.com/ask/answers/050515/it-better-use-fundamental-analysis-technical-analysis-or-quantitative-analysis-evaluate-longterm.asp (letzter Zugriff 28.05.2022)
- Investopedia (2022): Financial Markets. https://www.investopedia.com/terms/f/financial-market.asp (letzter Zugriff 28.05.2022)
- Investopedia (2022): Stock Markets. https://www.investopedia.com/terms/s/stockmarket.asp (letzter Zugriff 28.05.2022)
- Investopedia (2022): Technical Analysis. https://www.investopedia.com/terms/t/technicalanalysis.asp (letzter Zugriff 28.05.2022)
- McCarthy, J. (2004): What is Artificial Intelligence?. https://borghese.di.unimi.it/Teaching/AdvancedIntelligentSystems/Old/IntelligentSystems_2008_2009/Old/IntelligentSystems_2005_2006/Documents/Symbolic/04_McCarthy_whatisai.pdf (letzter Zugriff 02.05.2022)
- medium.com (2021): Stock Prediction Using Machine Learning Algorithms. https://hayirici.medium.com/stock-price-prediction-using-machine-learning-algorithms-961e6dce74f2 (letzer Zugriff 8.05.2022)
- Mohri, M. et al. (2012): Foundations of Machine Learning. Cambridge: The MIT Press
- Russell, S. und Norvig, P. (2021): Artificial intelligence: a modern approach. 4th Edition. Hoboken: Pearson Education
- SAP: What is machine learning? https://www.sap.com/insights/what-is-machine-learning.html (letzer Zugriff 02.06.2022)

- Shen, J. und Shafiq, M.O. (2020): Short-term stock market price trend prediction using a comprehensive deep learning system. https://doi.org/10.1186/s40537-020-00333-6 (letzer Zugriff 8.05.2022)
- Shen, S, et al (o. J.): Stock Market Forecasting Using Machine Learning Algorithms. http://cs229.stanford.edu/proj2012/ShenJiangZhang-StockMarketForecastingusingMachineLearningAlgorithms.pdf (letzer Zugriff 03.06.2022)
- Sutton, R.S. und Barto, A.G. (2015): Reinforcement Learning: An introduction. 2. Auflage. Cambridge: The MIT Press
- Swift (o.J.): About Swift. https://www.swift.org/about/ (letzer Zugriff 29.05.2022)
- Taag, T. (2022): Was einen guten Datensatz ausmacht. In: We are Developers! Ausgabe Frühjahr 2022, S. 30-36
- TATA Consultancy Services: White Paper: Machine Learning in Capital Markets. https://www.tcs.com/content/dam/tcs/pdf/Industries/Banking%20and%20Financial%20Services/Machine%20Learning%20in%20Capital%20Markets.pdf (letzer Zugriff 30.04.2022)
- Turing, A.M. (1950): Computing Machinery and Intelligence. https://www.csee.umbc.edu/courses/471/papers/turing.pdf (letzter Zugriff: 15.05.2022)
- UC Berkeley (2020): What is Machine Learning (ML). https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/ (letzter Zugriff 28.05.2022)
- Vadapalli, P. (2021): Stock Market Prediction Using Machine Learning. https://www.upgrad.com/blog/stock-market-prediction-using-machine-learning (letzer Zugriff 30.05.2022)
- Verbraucherzentrale (2021): Niedrigzinsen: Wie soll man sein Geld heute noch anlegen?. https://www.verbraucherzentrale.de/wissen/geld-versicherungen/sparen-und-anlegen/niedrigzinsen-wie-soll-man-sein-geld-heute-noch-anlegen-11534 (letzter Zugriff 11.1.22)
- Wikipedia (2022): Artificial Intelligence. https://en.wikipedia.org/wiki/Artificial_intelligence (letzter Zugriff 22.05.2022)
- Wikipedia (2022): Artificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network (letzter Zugriff 22.05.2022)
- Wikipedia (2022): Bias-variance tradeoff. https://en.wikipedia.org/wiki/Bias–variance_tradeoff (letzter Zugriff 27.05.2022)
- Wikipedia (2022): Decision tree learning. https://en.wikipedia.org/wiki/Decision_tree_learning (letzter Zugriff 27.05.2022)
- Wikipedia (2022): Deep learning. https://en.wikipedia.org/wiki/Deep_learning (letzter Zugriff 22.05.2022)
- Wikipedia (2022): Gradient boosting. https://en.wikipedia.org/wiki/Gradient_boosting (letzter Zugriff 28.05.2022)

- Wikipedia (2022): Logistic regression. https://en.wikipedia.org/wiki/Logistic_Regression (letzter Zugriff 27.05.2022)
- Wikipedia (2022): Machine learning. https://en.wikipedia.org/w/index.php?title=Machine_learning (letzter Zugriff 30.05.2022)
- Wikipedia (2022): Mean squared error. https://en.wikipedia.org/wiki/Mean_squared_error (letzter Zugriff 27.05.2022)
- Wikipedia (2022): regression analysis. https://en.wikipedia.org/wiki/Regression_analysis (letzter Zugriff 22.05.2022)
- Wikipedia (2022): Reinforcement learning. https://en.wikipedia.org/wiki/Reinforcement_learning (letzter Zugriff 28.05.2022)
- Wikipedia (2022): Statistical learning theory. https://en.wikipedia.org/wiki/Statistical_learning_theory (letzter Zugriff 27.05.2022)
- Wikipedia (2022): Supervised learning. https://en.wikipedia.org/wiki/Supervised_learning (letzter Zugriff 22.05.2022)
- Wikipedia (2022): Support-vector machine. https://en.wikipedia.org/wiki/Support-vector_machine (letzter Zugriff 27.05.2022)
- Yildirim, S. (2020): Gradient-Boosted Decision Trees-Explained. https://towardsdatascience.com/gradient-boosted-decision-trees-explained-9259bd8205af (letzter Zugriff 29.05.2020)
- Zhang, Y. und Dai Y. (o. J.): Machine Learning in Stock Price Trend Forecasting. http://cs229.stanford.edu/proj2013/DaiZhang-MachineLearningInStockPriceTrendForecasting.pdf (letzer Zugriff 03.06.2022)

Aufgrund von fehlenden sinngleichen Übersetzungen und wissenschaftlicher Genauigkeit wurden in dieser Seminararbeit die in der Forschung verwendeten englischen Fachbegriffe genutzt.

7. Versicherung der selbstständigen Verfassung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass ich alle Stellen, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, durch Angaben der Quellen als Zitate oder Entlehnungen kenntlich gemacht habe.

Reutlingen, 03.06.2022 **Luca Meurer**

Anhang I: Code für TradingGenie

```swift
//  TradingGenieApp.swift
//  Shared
//
//  Created by Luca on 20.11.21.

import SwiftUI

@main
struct TradingGenieApp: App {
    @StateObject private var model = Model()

    var body: some Scene {
        WindowGroup {
            ContentView().environmentObject(model)
                .preferredColorScheme(.dark)
        }
    }
}


//  ContentView.swift
//  TradingGenie
//
//  Created by Luca on 08/02/2022.

import SwiftUI

struct ContentView: View {
    #if os(iOS)
    @Environment(\.horizontalSizeClass) var sizeClass
    #endif
    @EnvironmentObject var model: Model

    @ViewBuilder
    var body: some View {
        #if os(macOS)
        sidebar
        #else
        if sizeClass == .compact {
            tabnav
        } else {
            sidebar
        }
        #endif
    }

    var sidebar: some View {
        NavigationView {
            List {
                NavigationLink(destination: HomeView().environmentObject(model)) {
                        Label("Home", systemImage: "house.fill")

                }
                NavigationLink(destination: PredictionView().environmentObject(model)) {
                        Label("Prediction", systemImage: "sparkles")

                }
                NavigationLink(destination: PriceView().environmentObject(model)) {
                        Label("Historical Data", systemImage: "mail.stack.fill")

                }
            }.listStyle(SidebarListStyle())
                .toolbar {
                    Button {} label: {

                    }
                }
                .navigationTitle("TradingGenie")
        }.environmentObject(model)
```

```
    }

    var tabnav: some View {
        TabView {
            HomeView().environmentObject(model)
                .tabItem {
                    Label("Home", systemImage: "house")
                }
            PredictionView().environmentObject(model)
                .tabItem {
                    Label("Prediction", systemImage: "sparkles")
                }
            PriceView().environmentObject(model)
                .tabItem {
                    Label("Historical Data", systemImage: "mail.stack.fill")
                }
        }.environmentObject(model)
    }
}

struct ContentView_Previews: PreviewProvider {
    static let model = Model()
    static var previews: some View {
        ContentView().environmentObject(model)
    }
}

//  HomeView.swift
//  TradingGenie
//
//  Created by Luca on 08/02/2022.

import SwiftUI

struct HomeView: View {
    @EnvironmentObject var model: Model

    var body: some View {
        PlatformNavAdapt(title: "Home") {
            ScrollView {
                if !model.predictions.isEmpty {
                    ForEach(model.predictions) { Prediction in
                        PredictionOverview(modelPrediction: Prediction)
                    }

                }
            }.padding()
                .toolbar {}
            }
    }
}

struct HomeView_Previews: PreviewProvider {
    static let model = Model()
    static var previews: some View {
        HomeView().environmentObject(model)
    }
}


//  PredictionView.swift
//  Shared
//
//  Created by Luca on 20.11.21.

import SwiftUI

struct PredictionView: View {
    @EnvironmentObject var model: Model

    var body: some View {
        PlatformNavAdapt(title: "Prediction") {
            ScrollView {
```

```
                VStack(alignment: .leading) {
                    GroupBox {
                        HStack {
                            Text("Data Input")
                                .font(.system(.headline, design: .rounded))
                            Spacer()
                        }
                        VStack(alignment: .leading) {
                            FancyTextField(input: $model.predictionSymbol, txt: "Symbol")

                            Picker("Time Scope", selection: $model.predictionTimeType) {
                                Text("Next Day").tag(MLmodel.predictionType.nextDay)
                                Text("Next Week").tag(MLmodel.predictionType.nextWeek)
                                Text("Next Month").tag(MLmodel.predictionType.nextMonth)
                            }.pickerStyle(.segmented)
                        }
                    }
                    GroupBox {
                        FancyDisclosure(title: "Enter Details manually") {
                            HStack {
                                Text("Closing Price")
                                    .font(.subheadline)
                                Spacer()
                                Text("Trading Volume")
                                    .font(.subheadline)
                            }
                        }
                    }
                }
            }.padding()
            .safeAreaInset(edge: .bottom, spacing: 0) {
                    bottomBar
                }
            .toolbar {
                ExportButton().environmentObject(model)
            }
        }
    }
    var bottomBar: some View {
        VStack {
            Button {
                Task {
                    var newModel = MLmodel()
                    newModel.symbol = model.predictionSymbol
                    await newModel.getData()
                        newModel.compileTrainingData(newModel.historicData)
                    newModel.train()
                    await newModel.predict()
                    newModel.saveModels()
                    model.predictions.append(newModel)
                }

            } label: {
                HStack {
                    Spacer()
                    Label("Make Prediction...", systemImage: "chart.line.uptrend.xyaxis")
                    Spacer()
                }.padding(.vertical, 5)
        }.buttonStyle(BorderedProminentButtonStyle())
        .controlSize(.regular)
         .padding(.horizontal, 40)
         .padding(.vertical, 16)
        }
            .background(.bar)
    }
}

struct PredictionView_Previews: PreviewProvider {
    static let model = Model()
    static var previews: some View {
        PredictionView().environmentObject(model)
            .colorScheme(.dark)
    }
```

```
}


//  Prices.swift
//  TradingGenie
//
//  Created by Luca on 09/02/2022.

import SwiftUI

struct PriceView: View {
    @EnvironmentObject var model: Model

    var body: some View {
        PlatformNavAdapt(title: "Historical Prices") {
                ScrollViewReader { scrollView in
                    ScrollView {
                    GroupBox {
                        VStack(alignment: .leading) {
                            FancyTextField(input: $model.priceRequestSymbol, txt: "Symbol")
                            DatePicker("Start Date", selection: $model.pRDate1)
                            DatePicker("To Date", selection: $model.pRDate2)
                            Picker(selection: $model.priceRequestInterval, label:
Text("Interval")) {
                                Text("1 Hour").tag(timeFormats.oneh)
                                Text("1 Day").tag(timeFormats.oneday)
                                Text("1 Week").tag(timeFormats.oneweek)
                                Text("1 Month").tag(timeFormats.onemonth)
                            }.pickerStyle(.segmented)
                        }
                    } label: {
                        Text("Data Input")
                    }

                    Button {
                        Task {
                            model.historicDataRequest = await fetchData(timeFormat:
model.priceRequestInterval, symbol: model.priceRequestSymbol, time1: model.pRDate1, time2:
model.pRDate2)
                        }
                        scrollView.scrollTo("Chart", anchor: .top)
                    } label: {
                        HStack {
                            Spacer()
                            Label("Get Prices...", systemImage: "arrow.down.doc")
                            Spacer()
                        }
                    }
                    .buttonStyle(BorderedButtonStyle())
                    .controlSize(.large)
                    .padding()
                        ChartView(prices: model.historicDataRequest.prices).id("Chart")

                    Divider().id("Prices")
                        PricesView(priceData: model.historicDataRequest, model: model)
                    }.padding()
                }.toolbar {
                    ExportButton().environmentObject(model)
                }
                }

    }
}

struct PriceView_Previews: PreviewProvider {
    static let model = Model()
    static var previews: some View {
        PriceView().environmentObject(model)
            .colorScheme(.dark)
    }
}

//  Types.swift
```

```swift
//  TradingGenie
//
//  Created by Luca on 12/02/2022.

import Foundation

var intervals = ["1m", "2m", "5m", "15m", "30m", "60m", "90m", "1h", "1d", "5d", "1wk", "1mo",
"3mo"]

enum timeFormats: String {
    case halfh = "30min"
    case oneh = "60min"
    case oneday = "1d"
    case oneweek = "1wk"
    case onemonth = "1mo"
}

enum FetchError: Error {
       case badRequest
       case badJSON
   }

//
//  Extensions.swift
//  TradingGenie
//
//  Created by Luca on 31/05/2022.
//

import Foundation

let yearFormatter: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy"
    return formatter
}()

let percentageFomatter: NumberFormatter = {
    let formatter = NumberFormatter()
    formatter.numberStyle = .percent
    formatter.minimumFractionDigits = 1
    formatter.maximumFractionDigits = 3
    formatter.alwaysShowsDecimalSeparator = true
    formatter.negativePrefix = "-"
    formatter.positivePrefix = "+"
    return formatter
}()

let stockFomatter: NumberFormatter = {
    let formatter = NumberFormatter()
    formatter.numberStyle = .decimal
    formatter.negativePrefix = "-"
    formatter.positivePrefix = "+"
    return formatter
}()

let stockPriceFomatter: NumberFormatter = {
    let formatter = NumberFormatter()
    formatter.numberStyle = .decimal
    formatter.maximumFractionDigits = 2
    return formatter
}()

extension Date {
    var dayInYear: Int {
        let cal = Calendar.current
        return cal.ordinality(of: .day, in: .year, for: self) ?? 0
    }
}

// Just for day
let AlVaDayFormatter: DateFormatter = {
    let formatter = DateFormatter()
```

```
    formatter.dateFormat = "yyyy-MM-dd"
    return formatter
}()
// For Day and Time
let AlVaDateFormatter: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy-MM-dd' 'HH:mm:ss"
    return formatter
}()

extension URLSession {
    func decodeDat<T: Decodable>(
        _ type: T.Type = T.self,
        from url: URL,
        keyDecodingStrategy: JSONDecoder.KeyDecodingStrategy = .useDefaultKeys,
        dataDecodingStrategy: JSONDecoder.DataDecodingStrategy = .deferredToData,
        dateDecodingStrategy: JSONDecoder.DateDecodingStrategy = .deferredToDate
    ) async throws  -> T {
        let (data, _) = try await data(from: url)

        let decoder = JSONDecoder()
        decoder.keyDecodingStrategy = keyDecodingStrategy
        decoder.dataDecodingStrategy = dataDecodingStrategy
        decoder.dateDecodingStrategy = dateDecodingStrategy

        let decoded = try decoder.decode(T.self, from: data)
        return decoded
    }
}

//  ML.swift
//  TradingGenie
//
//  Created by Luca on 05.12.21.

import Foundation
import CoreML
#if os(macOS)
import CreateML
#endif

struct MLmodel: Identifiable {
    var id: UUID = UUID()
    var symbol: String = "AAPL"
    var type: predictionType = .nextDay

    var historicData = historicPriceData()

    enum predictionType: String {
        case nextDay = "next-day"
        case nextWeek = "next-week"
        case nextMonth = "next-month"

    }

    mutating func getData() async {
        historicData = await fetchData(timeFormat: .oneday, symbol: symbol, time1:
Calendar.current.date(byAdding: .day, value: -20, to: Date.now) ?? Date(), time2: Date.now)

            date = Date()
            t1data = historicData.prices[0]
            t2data = historicData.prices[1]
            t3data = historicData.prices[2]
            t4data = historicData.prices[3]
            t5data = historicData.prices[4]
    }

        var date = Calendar.current.date(byAdding: .day, value: 1, to: Date.now) ?? Date()
        var predictedClose: Double = 14520
        var predictedTrendUp = true
        var t1data = intervalData()
        var t2data = intervalData()
        var t3data = intervalData()
```

```
        var t4data = intervalData()
        var t5data = intervalData()

        var isGain: Bool {
            predictedClose >= t1data.close
        }
        var difference: Double {
            predictedClose - t1data.close
        }
        var differencePercent: Double {
            difference/t1data.close
        }

        mutating func predict() async {
            let predictionData = try! MLDataTable(namedColumns: ["dayInYear":
MLUntypedColumn([date.dayInYear]), "t1Close": MLUntypedColumn([t1data.close]), "t1Open":
MLUntypedColumn([t1data.open]), "t1High": MLUntypedColumn([t1data.high]), "t1Low":
MLUntypedColumn([t1data.low]), "t1Volume": MLUntypedColumn([t1data.volume]), "t2Close":
MLUntypedColumn([t2data.close]), "t2Open": MLUntypedColumn([t2data.open]), "t2High":
MLUntypedColumn([t2data.high]), "t2Low": MLUntypedColumn([t2data.low]), "t2Volume":
MLUntypedColumn([t2data.volume]), "t3Close": MLUntypedColumn([t3data.close]), "t3Open":
MLUntypedColumn([t3data.open]), "t3High": MLUntypedColumn([t3data.high]), "t3Low":
MLUntypedColumn([t3data.low]), "t3Volume": MLUntypedColumn([t3data.volume]), "t4Close":
MLUntypedColumn([t4data.close]), "t4Open": MLUntypedColumn([t4data.open]), "t4High":
MLUntypedColumn([t4data.high]), "t4Low": MLUntypedColumn([t4data.low]), "t4Volume":
MLUntypedColumn([t4data.volume]), "t5Close": MLUntypedColumn([t5data.close]), "t5Open":
MLUntypedColumn([t5data.open]), "t5High": MLUntypedColumn([t5data.high]), "t5Low":
MLUntypedColumn([t5data.low]), "t5Volume": MLUntypedColumn([t5data.volume])])

            let pricePrediction = try! pricePredictor?.predictions(from: predictionData)
            let trendPrediction = try! trendPredictor?.predictions(from: predictionData)

            predictedClose = pricePrediction?.doubles?.element(at: 0) ?? 0
            if (trendPrediction?.ints?.element(at: 0) == 1) {
                predictedTrendUp = true
            } else {
                predictedTrendUp = false
            }
        }

    #if os(macOS)

        var data = MLDataTable()
        var priceMSE = Double()
        var priceMaxError = Double()
        var ClassificationError = Double()

        var pricePredictor: MLLinearRegressor? = nil
        var trendPredictor: MLBoostedTreeClassifier? = nil

        mutating func compileTrainingData(_ histdata: historicPriceData) {
            var dayInYear: [Int] = []
            var trendUps: [Bool] = []
            var actualCloses: [Double] = []
            var t1closes:[Double] = []
            var t1opens:[Double] = []
            var t1highs:[Double] = []
            var t1lows: [Double] = []
            var t1volumes: [Int] = []
            var t2closes:[Double] = []
            var t2opens:[Double] = []
            var t2highs:[Double] = []
            var t2lows:[Double] = []
            var t2volumes: [Int] = []
            var t3closes:[Double] = []
            var t3opens:[Double] = []
            var t3highs:[Double] = []
            var t3lows:[Double] = []
            var t3volumes: [Int] = []
            var t4closes:[Double] = []
            var t4opens:[Double] = []
            var t4highs:[Double] = []
            var t4lows:[Double] = []
```

```
            var t4volumes: [Int] = []
            var t5closes:[Double] = []
            var t5opens:[Double] = []
            var t5highs:[Double] = []
            var t5lows:[Double] = []
            var t5volumes: [Int] = []
            for i in 0...(histdata.prices.count-6) {
                dayInYear.append(histdata.prices[i].date.dayInYear)
                trendUps.append(histdata.prices[i].isGain)
                actualCloses.append(histdata.prices[i].close)
                t1closes.append(histdata.prices[i+1].close)
                t1opens.append(histdata.prices[i+1].open)
                t1highs.append(histdata.prices[i+1].high)s
                t1lows.append(histdata.prices[i+1].low)
                t1volumes.append(histdata.prices[i+1].volume)
                t2closes.append(histdata.prices[i+2].close)
                t2opens.append(histdata.prices[i+2].open)
                t2highs.append(histdata.prices[i+2].high)
                t2lows.append(histdata.prices[i+2].low)
                t2volumes.append(histdata.prices[i+2].volume)
                t3closes.append(histdata.prices[i+3].close)
                t3opens.append(histdata.prices[i+3].open)
                t3highs.append(histdata.prices[i+3].high)
                t3lows.append(histdata.prices[i+3].low)
                t3volumes.append(histdata.prices[i+4].volume)
                t4closes.append(histdata.prices[i+4].close)
                t4opens.append(histdata.prices[i+4].open)
                t4highs.append(histdata.prices[i+4].high)
                t4lows.append(histdata.prices[i+4].low)
                t4volumes.append(histdata.prices[i+4].volume)
                t5closes.append(histdata.prices[i+5].close)
                t5opens.append(histdata.prices[i+5].open)
                t5highs.append(histdata.prices[i+5].high)
                t5lows.append(histdata.prices[i+5].low)
                t5volumes.append(histdata.prices[i+5].volume)
            }
            data = try! MLDataTable(namedColumns: ["dayInYear": MLUntypedColumn(dayInYear),
"trendUp": MLUntypedColumn(trendUps), "actualClose": MLUntypedColumn(actualCloses), "t1Close":
MLUntypedColumn(t1closes), "t1Open": MLUntypedColumn(t1opens), "t1High":
MLUntypedColumn(t1highs), "t1Low": MLUntypedColumn(t1lows), "t1Volume":
MLUntypedColumn(t1volumes), "t2Close": MLUntypedColumn(t2closes), "t2Open":
MLUntypedColumn(t2opens), "t2High": MLUntypedColumn(t2highs), "t2Low": MLUntypedColumn(t2lows),
"t2Volume": MLUntypedColumn(t2volumes), "t3Close": MLUntypedColumn(t3closes), "t3Open":
MLUntypedColumn(t3opens), "t3High": MLUntypedColumn(t3highs), "t3Low": MLUntypedColumn(t3lows),
"t3Volume": MLUntypedColumn(t3volumes), "t4Close": MLUntypedColumn(t4closes), "t4Open":
MLUntypedColumn(t4opens), "t4High": MLUntypedColumn(t4highs), "t4Low": MLUntypedColumn(t4lows),
"t4Volume": MLUntypedColumn(t4volumes), "t5Close": MLUntypedColumn(t5closes), "t5Open":
MLUntypedColumn(t5opens), "t5High": MLUntypedColumn(t5highs), "t5Low": MLUntypedColumn(t5lows),
"t5Volume": MLUntypedColumn(t5volumes)])

        }

        mutating func train() {
            let (trainingData, testingData) = data.randomSplit(by: 0.8)

            let priceParams = MLLinearRegressor.ModelParameters(validation:
MLLinearRegressor.ModelParameters.ValidationData.split(strategy: .automatic), maxIterations:
2000)
            pricePredictor = try? MLLinearRegressor(trainingData: trainingData, targetColumn:
"actualClose", featureColumns: ["dayInYear", "t1Close", "t1Open", "t1High", "t1Low", "t1Volume",
"t2Close", "t2Open", "t2High", "t2Low", "t2Volume", "t3Close", "t3Open", "t3High", "t3Low",
"t3Volume", "t4Close", "t4Open", "t4High", "t4Low", "t4Volume", "t5Close", "t5Open", "t5High",
"t5Low", "t5Volume"], parameters: priceParams)

            // For Testing Best Algorithm:
            // pricePredictor = try? MLRegressor(trainingData: trainingData, targetColumn:
"actualClose", featureColumns: ["dayInYear", "t1Close", "t1Open", "t1High", "t1Low", "t1Volume",
"t2Close", "t2Open", "t2High", "t2Low", "t2Volume", "t3Close", "t3Open", "t3High", "t3Low",
"t3Volume", "t4Close", "t4Open", "t4High", "t4Low", "t4Volume", "t5Close", "t5Open", "t5High",
"t5Low", "t5Volume"])

            let trendParams = MLBoostedTreeClassifier.ModelParameters(maxIterations: 2000)
```

```
            trendPredictor = try? MLBoostedTreeClassifier(trainingData: trainingData,
targetColumn: "trendUp", featureColumns: ["dayInYear", "t1Close", "t1Open", "t1High", "t1Low",
"t1Volume", "t2Close", "t2Open", "t2High", "t2Low", "t2Volume", "t3Close", "t3Open", "t3High",
"t3Low", "t3Volume", "t4Close", "t4Open", "t4High", "t4Low", "t4Volume", "t5Close", "t5Open",
"t5High", "t5Low", "t5Volume"], parameters: trendParams)

            // For Testing Best Algorithm:
            // trendPredictor = try? MLClassifier(trainingData: trainingData, targetColumn:
"trendUp", featureColumns: ["dayInYear", "t1Close", "t1Open", "t1High", "t1Low", "t1Volume",
"t2Close", "t2Open", "t2High", "t2Low", "t2Volume", "t3Close", "t3Open", "t3High", "t3Low",
"t3Volume", "t4Close", "t4Open", "t4High", "t4Low", "t4Volume", "t5Close", "t5Open", "t5High",
"t5Low", "t5Volume"])

            let priceEvaluation = pricePredictor?.evaluation(on: testingData)
            let trendEvaluation = trendPredictor?.evaluation(on: testingData)
            priceMSE = priceEvaluation?.rootMeanSquaredError ?? 0
            priceMaxError = priceEvaluation?.maximumError ?? 0

            ClassificationError = trendEvaluation?.ClassificationError ?? 0

            print(priceMSE)
            print(priceMaxError)
            print(ClassificationError)
        }

    func saveModels() {
        do {
            let homePath = FileManager.default.homeDirectoryForCurrentUser
            let documentsPath = homePath.appendingPathComponent("Documents/TradingGenie")

        let regressorMetadata = MLModelMetadata(author: "Luca Meurer",
                                                shortDescription: "Predicts the stock price.",
                                                version: "1.0")
            try pricePredictor!.write(to: documentsPath.appendingPathComponent("PricePredictor-\
(symbol)_\(id).mlmodel"),
                                metadata: regressorMetadata)

        let classifierMetadata = MLModelMetadata(author: "Luca Meurer",
                                                shortDescription: "Predicts the stock trend.",
                                                version: "1.0")
            try trendPredictor!.write(to: documentsPath.appendingPathComponent("TrendPredictor-\
(symbol)_\(id).mlmodel"),
                                metadata: classifierMetadata)
        } catch {

        }
    }
    #endif
}


//  RequestModel.swift
//  TradingGenie
//
//  Created by Luca on 05.12.21.

import Foundation

// MARK: - Welcome
struct RequestedData: Codable {
    let metaData: MetaData?
    let timeSeriesDaily: [String: TimeSeriesDaily]?

    enum CodingKeys: String, CodingKey {
        case metaData = "Meta Data"
        case timeSeriesDaily = "Time Series (Daily)"
    }
}

// MARK: - MetaData
struct MetaData: Codable {
    let the1Information, the2Symbol, the3LastRefreshed, the4OutputSize: String?
    let the5TimeZone: String?
```

```
    enum CodingKeys: String, CodingKey {
        case the1Information = "1. Information"
        case the2Symbol = "2. Symbol"
        case the3LastRefreshed = "3. Last Refreshed"
        case the4OutputSize = "4. Output Size"
        case the5TimeZone = "5. Time Zone"
    }
}

// MARK: - TimeSeriesDaily
struct TimeSeriesDaily: Codable {
    let the1Open, the2High, the3Low, the4Close: String?
    let the5Volume: String?

    enum CodingKeys: String, CodingKey {
        case the1Open = "1. open"
        case the2High = "2. high"
        case the3Low = "3. low"
        case the4Close = "4. close"
        case the5Volume = "5. volume"
    }
}

let sampleRequestData = RequestedData(metaData: MetaData(the1Information: "", the2Symbol: "",
the3LastRefreshed: "", the4OutputSize: "", the5TimeZone: ""), timeSeriesDaily: [:])


//  DataModel.swift
//  TradingGenie
//
//  Created by Luca on 30/01/2022.

import Foundation

class Model: ObservableObject {

    // MARK: Historic Price Request
    @Published var historicDataRequest = historicPriceData() {
        didSet {
                if let encoded = try? JSONEncoder().encode(historicDataRequest) {
                    UserDefaults.standard.set(encoded, forKey: "historicDataRequest")
                }
            }
    }
    init() {
        if let savedItems = UserDefaults.standard.data(forKey: "historicDataRequest") {
            if let decodedItems = try? JSONDecoder().decode(historicPriceData.self, from:
savedItems) {
                historicDataRequest = decodedItems
                return
            }
        }
        historicDataRequest = historicPriceData()
    }

    @Published var priceRequestSymbol: String = "AAPL"
    @Published var priceRequestInterval: timeFormats = .oneday
    @Published var pRDate1: Date = Calendar.current.date(from: DateComponents(year: 2022, month:
1)) ?? Date.distantPast
    @Published var pRDate2: Date = Date.now

    // MARK: Prediction
    @Published var predictions = [MLmodel]()
    @Published var predictionSymbol: String = "AAPL"
    @Published var predReqDate1: Date = Calendar.current.date(from: DateComponents(year: 2022,
month: 1)) ?? Date.distantPast
    @Published var predReqDate2: Date = Date.now
    @Published var predictionTimeType: MLmodel.predictionType = .nextDay
}

struct historicPriceData: Codable {
    // Metadata
```

```
    var symbol: String = ""
    var lastRefreshed: Date = Date()
    var timezone: String = ""
    var information: String = ""
    var timeInterval: String = ""
    var prices: [intervalData] = [intervalData]()
}

struct intervalData: Codable {
    var date: Date = Date()
    var low: Double = 0
    var volume: Int = 0
    var close: Double = 0
    var open: Double = 0
    var high: Double = 0

    // Calculated Properties
    var difference: Double { close - open }
    var differencePercentage: Double { (difference / open) * 100 }
    var isGain: Bool {
        return difference >= 0 ? true : false
    }

}
let sampleData = historicPriceData()
let sampleIntervals = [intervalData(date: Date(), low: 177.25999450683594, volume: 64062300,
close: 177.57000732421875, open: 178.08999633789062, high: 179.22999572753906),
intervalData(date: Date(), low: 177.7100067138672, volume: 104487900, close: 182.00999450683594,
open: 177.8300018310547, high: 182.8800048828125), intervalData(date: Date(), low:
179.1199951171875, volume: 99310400, close: 179.6999969482422, open: 182.6300048828125, high:
182.94000244140625), intervalData(date: Date(), low: 174.63999938964844, volume: 94537600,
close: 174.9199981689453, open: 179.61000061035156, high: 180.1699981689453), intervalData(date:
Date(), low: 171.63999938964844, volume: 96904000, close: 172.0, open: 172.6999969482422, high:
175.3000030517578), intervalData(date: Date(), low: 171.02999877929688, volume: 86580100, close:
172.1699981689453, open: 172.88999938964844, high: 174.13999938964844), intervalData(date:
Date(), low: 168.1699981689453, volume: 106765600, close: 172.19000244140625, open:
169.0800018310547, high: 172.5), intervalData(date: Date(), low: 170.82000732421875, volume:
76138300, close: 175.0800018310547, open: 172.32000732421875, high: 175.17999267578125),
intervalData(date: Date(), low: 174.82000732421875, volume: 74805200, close: 175.52999877929688,
open: 176.1199951171875, high: 177.17999267578125)]
```

```
//  FetchRequest.swift
//  TradingGenie
//
//  Created by Luca on 31/05/2022.

import Foundation
    func fetchData(timeFormat interval: timeFormats, symbol: String, time1: Date, time2: Date?)
async -> historicPriceData {
        var fetchResults = sampleRequestData
        let now = Date()
        let date1: Int = Int(time1.timeIntervalSince1970)
        let date2: Int = Int(time2?.timeIntervalSince1970 ?? now.timeIntervalSince1970)
        let api = APIController()

        var urlString = "https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=\
(symbol)&apikey=\(api.getKey())"

        switch interval {
        case .halfh:
            urlString = "https://www.alphavantage.co/query?
function=TIME_SERIES_INTRADAY&symbol=\(symbol)&interval=\(interval)&apikey=\(api.getKey())"
        case .oneh:
            urlString = "https://www.alphavantage.co/query?
function=TIME_SERIES_INTRADAY&symbol=\(symbol)&interval=\(interval)&apikey=\(api.getKey())"
        case .oneday:
```

```
            urlString = "https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=\
(symbol)&apikey=\(api.getKey())"
        case .oneweek:
            urlString = "https://www.alphavantage.co/query?
function=TIME_SERIES_WEEKLY_ADJUSTED&symbol=\(symbol)&apikey=\(api.getKey())"
        case .onemonth:
            urlString = "https://www.alphavantage.co/query?
function=TIME_SERIES_MONTHLY_ADJUSTED&symbol=\(symbol)&apikey=\(api.getKey())"

        }

        print(urlString)

        do {
            let (data, response) = try await URLSession.shared.data(for: URLRequest(url:
URL(string: urlString)!))
                    guard (response as? HTTPURLResponse)?.statusCode == 200 else { throw
FetchError.badRequest }
            print("downloaded")
            // we got some data back!

            let decoder = JSONDecoder()
                if let decoded = try? decoder.decode(RequestedData.self, from: data) {
                   fetchResults = decoded
                }
            // success — convert the array values to our pages array
        } catch {

        }
        let data = await convertData(fetchResults)
        return data
    }

    func convertData(_ input: RequestedData) async -> historicPriceData {
        print(input)
        var data = historicPriceData()
        // Metadata
        data.symbol = input.metaData?.the2Symbol ?? "Unknown Symbol"
        data.lastRefreshed = AlVaDateFormatter.date(from: input.metaData?.the3LastRefreshed ??
"2022-04-12") ?? Date()
        data.timezone = input.metaData?.the5TimeZone ?? ""
        data.information = input.metaData?.the1Information ?? ""
        // Interval Data
        var prices: [intervalData] = []
        for timeSeries in input.timeSeriesDaily ?? [:] {
            var interval = intervalData()
            interval.date = AlVaDayFormatter.date(from: timeSeries.key) ?? Date()
            interval.close = Double(timeSeries.value.the4Close ?? "167.6600") ?? 167.6600
            interval.open = Double(timeSeries.value.the1Open ?? "167.6600") ?? 167.6600
            interval.high = Double(timeSeries.value.the2High ?? "167.6600") ?? 167.6600
            interval.low = Double(timeSeries.value.the3Low ?? "167.6600") ?? 167.6600
            interval.volume = Int(timeSeries.value.the5Volume ?? "167") ?? 167
            prices.append(interval)
        }
        data.prices = prices.sorted {$0.date > $1.date}
        return data
    }



//  APIController.swift
//  TradingGenie
//
//  Created by Luca on 13/04/2022.


import Foundation
import SwiftUI

struct APIController {
    @AppStorage("apiIndex") var apiIndex = 0
```

```
    let keys: [String] =
["Y5G5JG0OX42PTWQM","CIDK5RNSDPUOWSBL","R5BZ728KKDAJ2SMS","XKN1TR8LZ9SV5OWI","J7Y0JQ6NJKSQJ2WW",
"XY6F9PRR4E97F9KZ"]

    func getKey() -> String {
        let key = keys[apiIndex]
        apiIndex += 1
        if apiIndex == keys.count {
            apiIndex = 0
        }
        return key
    }
}


//  TextField.swift
//  TradingGenie
//
//  Created by Luca on 25.11.21.

import SwiftUI

struct FancyTextField: View {
    @Binding var input: String
    var txt: String

    var body: some View {
        TextField(txt, text: $input)
            .textFieldStyle(PlainTextFieldStyle())
            .padding(5)
            .background(Color.primary.opacity(0.1))
            .clipShape(RoundedRectangle(cornerRadius: 5))
    }
}

struct TextField_Previews: PreviewProvider {
    static var previews: some View {
        FancyTextField(input: .constant("Hello"), txt: "")
    }
}

//  PricesView.swift
//  TradingGenie
//
//  Created by Luca on 30/01/2022.

import SwiftUI

let intervalDF: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateStyle = .short
    formatter.timeStyle = .short
    return formatter
}()

struct PricesView: View {
    var priceData: historicPriceData
    var model: Model

    var body: some View {
        VStack {
                HStack(alignment: .top) {
                    VStack(alignment: .leading) {
                        Text("Price History")
                            .font(.title2)
                            .bold()
                        Text(priceData.symbol)
                            .fontWeight(.semibold)
                    }
                    Spacer()
                    VStack(alignment: .trailing) {
                        Text("Time Zone: \(priceData.timezone )")
                            .font(.footnote)
```

```
                        Text("Interval: \(model.priceRequestInterval.rawValue)")
                            .font(.footnote)
                    }
                }
                    .padding(10)



                    ScrollView {
                        VStack(alignment: .leading, spacing: 0) {
                            Divider()
                            ForEach(priceData.prices, id: \.date) { intervalData in
                                HStack {
                                    Text(intervalDF.string(from: intervalData.date))
                                        .fontWeight(.medium)
                                        .foregroundStyle(.secondary)
                                    Spacer()
                                    Text(percentageFomatter.string(from:
intervalData.difference/intervalData.open as NSNumber)!)
                                        .font(.system(.caption, design: .monospaced))
                                        .foregroundColor(intervalData.isGain ? .green : .red)
                                    Text("\(String(format: "%.2f", intervalData.close))")
                                        .fontWeight(.semibold)

                                }.font(.footnote)
                                    .padding(.vertical, 4)
                                Divider()
                            }
                        }
                        .padding([.bottom, .horizontal], 10)
                    }

        }
        }
    }


struct PricesView_Previews: PreviewProvider {
    static var previews: some View {
        PricesView(priceData: sampleData, model: Model())
    }
}







//  PredictionOverview.swift
//  TradingGenie
//
//  Created by Luca on 06/02/2022.

import SwiftUI

struct PredictionOverview: View {
    var modelPrediction: MLmodel
    @State var showDetails: Bool = false

    var body: some View {
        ZStack {
                GroupBox {
                    ZStack(alignment: .bottomTrailing) {
                        HStack(alignment: .top) {
                                VStack(alignment: .leading) {
                                    Text(modelPrediction.symbol)
                                        .font(.system(.largeTitle, design: .rounded).bold())

                                    Text(modelPrediction.type.rawValue)
                                        .font(.system(.footnote, design: .monospaced))
                                    Text(stockPriceFomatter.string(from:
modelPrediction.predictedClose as NSNumber)!)
```

```
                                    .font(.system(.largeTitle, design: .rounded).bold())
                                Spacer()
                                HStack {
                                    Text(stockFomatter.string(from:
modelPrediction.difference as NSNumber)!)
                                        .font(.system(size: 20))
                                    Text(percentageFomatter.string(from:
modelPrediction.differencePercent as NSNumber) ?? "Nil")
                                        .font(.system(size: 20))
                                }.foregroundColor(modelPrediction.isGain ? .green : .red)
                            }
                            Spacer()
                            Image(systemName: "arrow.right.circle")
                                    .resizable()
                                    .foregroundColor(modelPrediction.predictedTrendUp ? .gre
en : .red)
                                    .scaledToFit()
                                    .rotationEffect(Angle(degrees:
modelPrediction.predictedTrendUp ? 270 : 90))
                                    .frame(maxWidth: 120)
                        }
                        Button {
                            withAnimation {
                                showDetails.toggle()
                            }
                        } label: {
                            Image(systemName: "info.circle.fill")
                                .resizable()
                                .scaledToFit()
                                .foregroundColor(.accentColor)
                                .frame(maxHeight: 30)
                        }.buttonStyle(SquishableButtonStyle(fadeOnPress: true))
                    }.frame(minHeight: 150, maxHeight: 150)
                    if showDetails {
                        Text("Trend Probability: \((1-
modelPrediction.ClassificationError)*100)%")
                        Text("Mean Square Error: \(modelPrediction.priceMSE)")
                    }
                }

        }
    }
}

struct PredictionOverview_Previews: PreviewProvider {
    static let model = MLmodel()
    static var previews: some View {
        PredictionOverview(modelPrediction: model)
            .preferredColorScheme(.dark)
    }
}

struct SquishableButtonStyle: ButtonStyle {
    var fadeOnPress = true

    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .opacity(configuration.isPressed && fadeOnPress ? 0.75 : 1)
            .scaleEffect(configuration.isPressed ? 0.95 : 1)
    }
}

extension ButtonStyle where Self == SquishableButtonStyle {
    static var squishable: SquishableButtonStyle {
        SquishableButtonStyle()
    }

    static func squishable(fadeOnPress: Bool = true) -> SquishableButtonStyle {
        SquishableButtonStyle(fadeOnPress: fadeOnPress)
    }
}
```

```swift
//  PlatformNavAdapt.swift
//  TradingGenie
//
//  Created by Luca on 10/02/2022.

import SwiftUI

struct PlatformNavAdapt<Content: View>: View {
    var title: String?

    @ViewBuilder var input: () -> Content
    #if os(iOS)
    @Environment(\.horizontalSizeClass) var sizeClass
    #endif

    var body: some View {
            #if os(macOS)
            ScrollView {
                ZStack(alignment: .topTrailing) {
                    ZStack(alignment: .bottomLeading) {
                        LinearGradient(gradient: Gradient(colors: [.black, .accentColor]),
startPoint: .topLeading, endPoint: .bottomTrailing)

                        Text(title ?? "")
                            .padding(.all)
                            .font(.system(.title, design: .rounded))
                            .foregroundColor(.white)
                    }
                }
                input()
                    .padding()
            }
            #else
            if sizeClass == .compact {
                NavigationView {
                    input()
                        .navigationTitle(title ?? "")
                }
            } else {
                input()
                    .navigationTitle(title ?? "")
            }
            #endif
    }
}

struct PlatformNavAdapt_Previews: PreviewProvider {
    static var previews: some View {
        PlatformNavAdapt(title: "") {
            Text("Hello")
        }
    }
}
```

Anhang II: Ausgabewerte

Training zur Ermittlung des besten Algorithmus:

```
Using 26 features to train a model to predict actualClose.

Linear regression:
--------------------------------------------------------
Number of examples          : 74
Number of features          : 26
Number of unpacked features : 26
Number of coefficients    : 27
Starting Newton Method
--------------------------------------------------------
+-----------+----------+--------------+--------------------+----------------------
+----------------------------------+-----------------------------------+
| Iteration | Passes   | Elapsed Time | Training Max Error | Validation Max Error | Training Root-Mean-
Square Error | Validation Root-Mean-Square Error | Offset
+-----------+----------+--------------+--------------------+----------------------
+----------------------------------+-----------------------------------+
| 1         | 2        | 0.002134     | 8.413112           | 10.112931            | 3.228298
| 4.764570                         |   0.03024
+-----------+----------+--------------+--------------------+----------------------
+----------------------------------+-----------------------------------+
SUCCESS: Optimal solution found.

Boosted trees regression:
--------------------------------------------------------
Number of examples          : 74
Number of features          : 26
Number of unpacked features : 26
+-----------+--------------+--------------------+----------------------
+----------------------------------+-----------------------------------+
| Iteration | Elapsed Time | Training Max Error | Validation Max Error | Training Root-Mean-Square Error
| Validation Root-Mean-Square Error |
+-----------+--------------+--------------------+----------------------
+----------------------------------+-----------------------------------+
| 1         | 0.002451     | 130.618164         | 129.428162           | 114.256302
| 117.743439                       |
| 2         | 0.003298     | 96.937515          | 95.747513            | 80.772003
| 84.193512                        |
| 3         | 0.004154     | 72.216461          | 71.026459            | 57.272877
| 60.293697                        |
| 4         | 0.005055     | 54.333633          | 53.143631            | 40.712055
| 43.446953                        |
| 5         | 0.005898     | 40.420547          | 39.230545            | 29.000137
| 31.196123                        |
| 10        | 0.011352     | 13.237427          | 11.715469            | 5.896276
| 7.795638                         |
+-----+--------------+--------------------+----------------------+--------------------------------
+----------------------------------+

Using 26 features to train a model to predict trendUp.

Boosted trees classifier:
--------------------------------------------------------
Number of examples          : 74
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
+-----------+--------------+-------------------+--------------------+------------------
+---------------------+
| Iteration | Elapsed Time | Training Accuracy | Validation Accuracy | Training Log Loss | Validation
Log Loss |
+-----------+--------------+-------------------+--------------------+------------------
+---------------------+
| 1         | 0.003103     | 0.986486          | 0.875000           | 0.536992          | 0.579093
|
| 2         | 0.004808     | 0.986486          | 0.750000           | 0.417587          | 0.597589
|
```

```
| 3         | 0.006489     | 0.986486          | 0.750000            | 0.335526          | 0.584928
|
| 4         | 0.008249     | 1.000000          | 0.500000            | 0.272861          | 0.636027
|
| 5         | 0.010078     | 1.000000          | 0.750000            | 0.226648          | 0.603921
|
| 10        | 0.018638     | 1.000000          | 0.750000            | 0.112836          | 0.544072
|
+-----------+--------------+-------------------+---------------------+-------------------
Random forest classifier:
--------------------------------------------------------
Number of examples          : 74
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
+-----------+--------------+-------------------+---------------------+-------------------
| Iteration | Elapsed Time | Training Accuracy | Validation Accuracy | Training Log Loss | Validation
Log Loss |

| 1         | 0.003214     | 0.891892          | 0.500000            | 0.411672          | 0.929326
|
| 2         | 0.004865     | 0.918919          | 0.500000            | 0.364168          | 0.718018
|
| 3         | 0.006588     | 0.972973          | 0.750000            | 0.341957          | 0.638254
|
| 4         | 0.008119     | 0.972973          | 0.625000            | 0.341602          | 0.628935
|
| 5         | 0.009731     | 0.972973          | 0.750000            | 0.338667          | 0.605486
|
| 10        | 0.016851     | 1.000000          | 0.625000            | 0.341076          | 0.604678
|
+-----------+--------------+-------------------+---------------------+-------------------
Decision tree classifier:
--------------------------------------------------------
Number of examples          : 74
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
+-----------+--------------+-------------------+---------------------+-------------------
+----------------------+
| Iteration | Elapsed Time | Training Accuracy | Validation Accuracy | Training Log Loss | Validation
Log Loss |
+-----------+--------------+-------------------+---------------------+-------------------
+----------------------+
| 1         | 0.003622     | 0.986486          | 0.65000             | 0.296446          | 0.432015
|
+-----------+--------------+-------------------+---------------------+-------------------
SVM:
--------------------------------------------------------
Number of examples          : 74
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
Number of coefficients    : 27
Starting L-BFGS
--------------------------------------------------------
+-----------+----------+-----------+--------------+-------------------+----------------------+
| Iteration | Passes   | Step size | Elapsed Time | Training Accuracy | Validation Accuracy |
+-----------+----------+-----------+--------------+-------------------+----------------------+
| 0         | 6        | 0.666271  | 0.001047     | 0.527027          | 0.375000             |
| 1         | 11       | 0.033618  | 0.001956     | 0.527027          | 0.375000             |
| 2         | 12       | 0.042023  | 0.002418     | 0.527027          | 0.375000             |
| 3         | 16       | 0.882477  | 0.003304     | 0.527027          | 0.375000             |
| 4         | 17       | 1.000000  | 0.003792     | 0.527027          | 0.375000             |
| 9         | 25       | 1.000000  | 0.006153     | 0.554054          | 0.625000             |
+-----------+----------+-----------+--------------+-------------------+----------------------+
Logistic regression:
--------------------------------------------------------
Number of examples          : 74
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
Number of coefficients      : 27
```

```
Starting Newton Method
--------------------------------------------------------
+-----------+----------+--------------+-------------------+----------------------+
| Iteration | Passes   | Elapsed Time | Training Accuracy | Validation Accuracy |
+-----------+----------+--------------+-------------------+----------------------+
| 1         | 2        | 0.000661     | 0.608108          | 0.625000            |
| 2         | 3        | 0.001147     | 0.608108          | 0.625000            |
+-----------+----------+--------------+-------------------+----------------------+
SUCCESS: Optimal solution found.
```

Training für AAPL:

```
Using 26 features to train a model to predict actualClose.

Linear regression:
--------------------------------------------------------
Number of examples          : 82
Number of features          : 26
Number of unpacked features : 26
Number of coefficients    : 27
Starting Newton Method
--------------------------------------------------------
+-----------+----------+--------------+-------------------+----------------------
+----------------------------------+----------------------------------+
| Iteration | Passes   | Elapsed Time | Training Max Error | Validation Max Error | Training Root-Mean-
Square Error | Validation Root-Mean-Square Error | Offset
+-----------+----------+--------------+-------------------+----------------------
+----------------------------------+----------------------------------+
| 1         | 2        | 0.000840     | 8.500871          | 7.056193             | 3.330355
| 3.889119                         |   0.02307
+-----------+----------+--------------+-------------------+----------------------
+----------------------------------+----------------------------------+
SUCCESS: Optimal solution found.

Using 26 features to train a model to predict trendUp.

Boosted trees classifier:
--------------------------------------------------------
Number of examples          : 72
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
+-----------+--------------+-------------------+--------------------+-------------------
+----------------------+
| Iteration | Elapsed Time | Training Accuracy | Validation Accuracy | Training Log Loss | Validation
Log Loss |
+-----------+--------------+-------------------+--------------------+-------------------
| 1         | 0.002748     | 1.000000          | 0.600000           | 0.527664          | 0.658814
|
| 2         | 0.004526     | 1.000000          | 0.500000           | 0.418362          | 0.641726
|
| 3         | 0.006176     | 1.000000          | 0.700000           | 0.338519          | 0.602486
|
| 4         | 0.007920     | 1.000000          | 0.800000           | 0.282140          | 0.563348
|
| 5         | 0.009463     | 1.000000          | 0.700000           | 0.242496          | 0.564763
|
| 10        | 0.016829     | 1.000000          | 0.800000           | 0.121505          | 0.562738
|
| 50        | 0.068499     | 1.000000          | 0.700000           | 0.018313          | 0.671480
|
| 100       | 0.121901     | 1.000000          | 0.800000           | 0.010210          | 0.750081
|
| 500       | 0.472659     | 1.000000          | 0.800000           | 0.004600          | 0.850134
|
| 1000      | 0.843921     | 1.000000          | 0.800000           | 0.003815          | 0.874234
|
| 2000      | 1.662384     | 1.000000          | 0.800000           | 0.003139          | 0.894347
|
+-----+-------+-------------------+--------------------+-------------------
```

Training für AIR.DE:

```
Using 26 features to train a model to predict actualClose.

Linear regression:
--------------------------------------------------------
Number of examples          : 82
Number of features          : 26
Number of unpacked features : 26
Number of coefficients    : 27
Starting Newton Method
--------------------------------------------------------
+-----------+----------+--------------+--------------------+----------------------
+---------------------------------+-----------------------------------+
| Iteration | Passes   | Elapsed Time | Training Max Error | Validation Max Error | Training Root-Mean-
Square Error | Validation Root-Mean-Square Error | Offset
+-----------+----------+--------------+--------------------+----------------------
+---------------------------------+-----------------------------------+
| 1         | 2        | 0.001657     | 8.360145           | 4.068484             | 2.630614
| 1.983172                          |    0.01920
+-----------+----------+--------------+--------------------+----------------------
+---------------------------------+-----------------------------------+
SUCCESS: Optimal solution found.

Using 26 features to train a model to predict trendUp.

Boosted trees classifier:
--------------------------------------------------------
Number of examples          : 72
Number of classes           : 2
Number of feature columns   : 26
Number of unpacked features : 26
+-----------+--------------+-------------------+--------------------+------------------
+---------------------+
| Iteration | Elapsed Time | Training Accuracy | Validation Accuracy | Training Log Loss | Validation
Log Loss |
+-----------+--------------+-------------------+--------------------+------------------
+---------------------+
| 1         | 0.003062     | 0.986111          | 0.600000           | 0.523017          | 0.677848
|
| 2         | 0.004772     | 1.000000          | 0.600000           | 0.417194          | 0.660367
|
| 3         | 0.006535     | 1.000000          | 0.600000           | 0.334487          | 0.662147
|
| 4         | 0.008138     | 1.000000          | 0.600000           | 0.278227          | 0.662742
|
| 5         | 0.009739     | 1.000000          | 0.600000           | 0.234068          | 0.667303
|
| 10        | 0.017603     | 1.000000          | 0.500000           | 0.113226          | 0.700977
|
| 50        | 0.069573     | 1.000000          | 0.600000           | 0.017896          | 0.814685
|
| 100       | 0.121939     | 1.000000          | 0.600000           | 0.009850          | 0.913780
|
| 500       | 0.438588     | 1.000000          | 0.700000           | 0.004387          | 0.895914
|
| 1000      | 0.775951     | 1.000000          | 0.700000           | 0.003705          | 0.941769
|
| 2000      | 1.458508     | 1.000000          | 0.700000           | 0.003107          | 1.109731
|
+-----------+--------------+-------------------+--------------------+------------------
```