Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

1.2 Ziel der Arbeit

Random Forests (Breiman, 2001) gehören zu den mächtigsten Machine Learning Algorithmen (Gron, 2017). In verschiedenen Domänen wurde ihre praktische Anwendbarkeit unter Beweis gestellt. Einige Beispiele sind Anwendungen in der Chemoinformatik (Svetnik u. a., 2003), Ökologie (Prasad u. a. 2006; Cutler u. a. 2007), 3D-Objekterkennung (Shotton u. a., 2011) und Bioinformatik (Diaz-Uriarte u. Alvarez, 2006) sowie ein Data Science Hackathon zur Luftqualitätsvorhersage (http://www.kaggle.com/c/dsg-hackathon) (Tang u. a., 2018). Howard u. Bowles (2012) bezeichnen Random Forests als den erfolgreichsten Allzweck-Algorithmus der neueren Zeit.

Derzeit sind Random Forests noch nicht ausreichend für die Finanzdomäne erforscht. Insbesondere die Wahl von Hyperparametern – die häufig willkürlich oder über Heuristiken gesetzt werden – stellt eine vielversprechende Forschungsfrage da. Eine Gegenüberstellung der Anwendbarkeit von von Random Forests für verschiedene Fragen, wie der Stock Price Prediction oder dem Credit Scoring, hat noch nicht stattgefunden.

Diese Arbeit hat das Ziel, die Anwendung von Decision Trees und Random Forests in der Finanzdomäne kritisch zu evaluieren und, wo sinnvoll, mittels Hyperparametern zu optimieren.

Ziele (DT=Decision Tree, RF=Random Forest):

- Forschungsfrage: Können DT und RF Klassifikatoren sinnvoll (besser als Dummy Classifier oder besser als Markt Index Performance) in der Finanzdomäne eingesetzt werden?
- Unterfrage 1: Können DT und RF sinnvoll für Credit Scoring (Anwendung 1) und Aktienempfehlungen (Anwendung 2) eingesetzt werden?
- Unterfrage 2: Wie sollten die Hyperparameter jeweils für DT und RF pro Anwendungsfall gesetzt werden?
- Unterfrage 2.5: Kann Pruning in Anwendung 1 und Anwendung 2 zur Vermeidung von Overfitting des DT beitragen?
- Unterfrage 3: Wie unterscheiden sich DT und RF Algorithmen bzw. wann eignet sich welcher Klassifikator?
- Unterfrage 3.5: Ist Meta Random Forest sinnvoll? Soft Voting vs Hard Voting?

- Unterfrage 4: Lassen sich relevante Effekte aus der realen Welt (Christmas Effekt, Dividendenausschüttung, Jahresabschluss-Veröffentlichung, etc.) in den Modellen wiederfinden?
- Unterfrage 5: Sind die Ergebnisse mit der Markteffizienzhypothese vereinbar?
- Unterfrage 6: Welche Einschränkungen gibt es bei der Anwendung von DT und RF Klassifikatoren in der Finanzdomäne?

Einschränkungen: Diese Arbeit untersucht nur binäre Klassifikation (keine Regression) und vernachlässigt sämtliche Transaktionskosten (da diese variabel sind etc.)

2 Theoretische Grundlagen

2.1 Machine Learning

Machine Learning bedeutet, einen Computer so zu programmieren, dass ein bestimmtes Leistungskriterium anhand von Beispieldaten oder Erfahrungswerten aus der Vergangenheit optimiert wird (Alpaydın, 2008). Diese Programme führen für ein gegebenen Input nicht immer zum selben Ergebnis, sondern lernen – ähnlich wie der Mensch – anhand von Beispielen. Deshalb eignet sich Machine Learning für Probleme, für die der Mensch keine simplen Regeln verfassen und in einem Algorithmus automatisieren kann. Stattdessen löst der Mensch solche Probleme anhand von Erfahrungen, teilweise anhand von Intuition. Ein Beispiel ist die Diagnose von Krebszellen. Fachärzte durchlaufen eine jahrelange Ausbildung und analysieren eine Vielzahl von Beispielbildern, um eine Einschätzung über neue Zellen treffen zu können. Eine solche Diagnose kann nicht mit einem simplen Algorithmus automatisiert werden, da jedes Zellenbild einzigartig ist. Es ist nicht realistisch, in einem Algorithmus alle Eventualitäten programmatisch abzudecken. Machine Learning hingegen ermöglicht es, den menschlichen Lernprozess nachzubilden und somit komplexe Diagnosen zu erstellen. Vorteile gegenüber dem Menschen sind dabei die Objektivität, da der Computer keinen Emotionen unterliegt, und die Fähigkeit, große Mengen an Daten in einem Bruchteil der Zeit, die ein Mensch benötigen würde, zu vergleichen.

Technisch gesehen lernt ein Machine Learning Programm, indem es eine Funktion

$$f(X|\Theta) = \hat{y}, \quad f \in F \tag{2.1}$$

aus dem Funktionenraum F bildet und durch Training mit Beispieldaten die Parameter Θ optimiert. Der Funktionenraum F umfasst alle Funktionen, die ein Modell erlernen kann, und ist vom gewählten Lernalgorithmus abhängig. Die Beispieldaten sind eine Stichprobe χ mit n Datensätzen in der Form

$$\chi = \{(X_1, y_1), ..., (X_n, y_n)\}. \tag{2.2}$$

Dabei steht X für einen m-dimensionalen Input-Vektor der Form

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \tag{2.3}$$

der als Informationsgrundlage zur Bestimmung von $\hat{y}$ dient. Die Variable $\hat{y}$ bezeichnet das Ergebnis der Funktion für einen gegebenen Input-Vektor X. Die Optimierung von $f(X|\Theta)$ bezieht sich hier auf die Suche jener Parameter Θ, welche die genaueste Näherung für die tatsächliche Funktion $f^*(X)$ und somit auch für die tatsächliche Klasse y erzielen (Alpaydın, 2008). Ist die Variable $\hat{y}$ numerisch, so spricht man von Regression. Ein Beispiel

für eine Regression ist die Vorhersage der Temperatur in Grad Celsius. Dabei enthält die Ergebnismenge der Funktion unendlich viele Elemente, zum Beispiel die reellen Zahlen. Ist die Variable $\hat{y}$ kategorisch, so spricht man von Klassifikation. Im Spezialfall einer binären Klassifikation gilt zudem

$$\hat{y} = \begin{cases} 1, & \text{wenn } X \text{ eine positive Instanz ist} \\ 0, & \text{wenn } X \text{ eine negative Instanz ist.} \end{cases} \tag{2.4}$$

Welche Instanzen positiv und welche negativ sind, ist vom Anwender zu definieren. Ein Anwendungsbeispiel für eine binäre Klassifikation ist die Bestimmung, ob eine gegebene Pflanze für den Menschen giftig ist oder nicht. Bei der Klassifikation ist die Menge der möglichen Werte von $\hat{y}$ endlich. In binären Fall kann $\hat{y}$, wie in Gleichung 2.4 gezeigt, genau zwei Werte annehmen.

Nach dem Training trifft das Modell Aussagen über Instanzen, die außerhalb von χ liegen. Die Generalisierungsfähigkeit des Modells mit Parametern Θ wird anhand der Abweichungen zwischen $\hat{y}$ und y für alle i Instanzen eines Validierungs-Datensets χ_{val} mittels einer Fehlerfunktion

$$E(\Theta|\chi_{val}) = \sum_i Diff(\hat{y}, y) \tag{2.5}$$

berechnet. Für die Diff-Funktion gibt es zahlreiche Methoden, deren Eignung vom konkreten Anwendungsfall abhängt (Alpaydın, 2008). Die in dieser Arbeit verwendeten Methoden sind in Kapitel **??**, "Methodik", erläutert. Das finale Modell verwendet anschließend jene Funktion, die die geringsten Abweichungen auf χ_{val} erreicht, also die beste Generalisierung aufweist. Um die erwartete Fehlerrate des finalen Modells zu bestimmen, werden die Abweichungen auf einem Test-Datenset χ_{test} – wie in Abschnitt 2.6 – herangezogen. Die Aufgabe von Machine Learning ist es also, jene Parameter Θ_{opt} zu finden, welche die Fehlerfunktion minimieren, also

$$\Theta_{opt} = \underset{\Theta}{\operatorname{argmin}}\, E(\Theta|\chi). \tag{2.6}$$

An dieser Stelle sei auf einen grundlegenden Trade-Off im Machine Learning hingewiesen: die Komplexität der Funktion, bedingt durch die Mächtigkeit des Funktionenraumes F, die Menge an Daten in χ und der Generalisierungsfehler sind voneinander abhängig (Alpaydın, 2008). Steigende Komplexität führt bei gleichbleibender Datenbasis zu einem höheren Generalisierungsfehler, kann aber bei größerer Datenbasis zu einem niedrigeren Generalisierungsfehler führen. Wenn die Datenmenge ceteris paribus steigt, nimmt der Generalisierungsfehler ab, und vice versa (Alpaydın, 2008).

Der Schwerpunkt dieser Arbeit liegt auf der Klassifikation. Deshalb wird diese in der folgenden Sektion genauer betrachtet.

2.2 Klassifikation

Klassifikation bezeichnet das Zuweisen einer Klasse $\hat{y}$ zu einem gegebenen Input-Vektor X. Ein Beispiel ist die Bestimmung der Kreditwürdigkeit eines Bankkunden, bekannt als das

Credit Scoring Problem (Abdou u. Pointon, 2011). Die Bank hat bei der Vergabe eines Kredites das Ziel, das Ausfallrisiko zu minimieren und den erwarteten Gewinn zu maximieren. Dazu werden von Kredit-Antragsstellern Datenpunkte wie Vermögen, Gehalt, Kredithistorie und Alter erhoben. Die gesammelten Daten dienen anschließend als Trainingsdaten für einen Machine Learning Algorithmus. Dieser erstellt ein Modell, das für die historischen Daten optimiert ist. Dieses Modell klassifiziert dann die Kreditwürdigkeit der Antragssteller. Dadurch wird die Bank bei der Kreditentscheidung unterstützt. In manchen Fällen wird die Entscheidung anhand von Schwellenwerten komplett automatisiert (Abdou u. Pointon, 2011).

Das vorangegangene Beispiel beinhaltet typische Elemente einer Klassifikation: Instanzen, Features und Klassen. Eine Instanz ist im Beispiel ein Kunde. Jede Instanz wird durch dieselben Attribute – aber womöglich mit unterschiedlichen Ausprägungen – beschrieben. Diese Attribute werden als Features, Φ, bezeichnet. Die Ausprägungen der Features für eine gegebene Instanz dienen als Input-Vektor X für die Klassifizierung, wie in Gleichung 2.1 dargestellt. Aufgrund der grundlegenden Bedeutung der Features stellen deren Berechnung (Feature Extraction) und Auswahl (Feature Selection) zentrale Schritte im Machine Learning dar, den das Kapitel **??** vertieft. Die möglichen kategorischen Werte der Ergebnis-Variable y, genannt Klassen, könnten im Beispielfall "Kunde mit hohem Risiko" und "Kunde mit geringem Risikoßein. Das Ergebnis einer Klassifikation ist die Klasse der betrachteten Instanz, im Beispiel die Kreditwürdigkeit eines Kunden.

Ein simpler Klassifikator könnte unter anderem anhand von folgender Regel handeln:

$$IF(\text{Salary} > 100.000 \wedge \text{Credit Amount} < 200.000), THEN\, Class = \text{"Low Risk"}. \tag{2.7}$$

Basierend auf dem Gehalt des Kunden und der Kreditsumme, schätzt das Modell das Risiko der Kreditvergabe ein. In der Realität reichen diese zwei Kriterien allerdings häufig nicht aus, um eine fundierte Entscheidung zu treffen. Bankangestellte prüfen ebenfalls die Kredithistorie, die Lebenssituation, den Arbeitsvertrag, den persönlichen Eindruck sowie weitere – möglicherweise auch die Persönlichkeit betreffende – Faktoren. Sofern diese Faktoren als Features in den Daten vorhanden sind, eignet sich der Klassifikator diese in der Trainingsphase an und ergänzt sie in das Entscheidungsmodell. Bis zu einem gewissen Punkt erhöht das Hinzufügen von Features und Regeln den Erfolg des Modells, wie in Abschnitt 2.7 dargestellt, bevor die zunehmende Komplexität die Generalisierungsfähigkeit vermindert.

Diese Bachelorarbeit beschränkt sich auf binäre Klassifikatoren, also Klassifikatoren mit genau zwei Klassen. Konkret werden Decision Tree und Random Forest Klassifikatoren sowie einige Abwandlungen dergleichen untersucht. Es folgen nun die theoretischen Grundlagen der Erstellung, Anwendung und Erfolgsmessung dieser Modelle.

2.3 Decision Tree Klassifikator

2.3.1 Generelle Funktionsweise und Eigenschaften

Die Induktion von Decision Trees (Quinlan, 1986) gehört zu den simpelsten aber erfolgreichsten Machine Learning Algorithmen (Russell u. Norvig, 2009). Gupta u. a. (2017) nennen unter anderem die medizinische Diagnostik, intelligente Fahrzeuge, das Credit Scoring (siehe

Abschnitt 2.2) und die industrielle Qualitätskontrolle als Anwendungsbereiche. Außerdem bilden Decision Trees die Grundlage für das später behandelte Random Forest Ensemble. Ein Decision Tree funktioniert nach dem Teile-und-Herrsche Prinzip. Die Trainings-Stichprobe $\chi_{training}$ wird nach einem festgelegten Kriterium in Teilmengen geteilt. Die entstandenen Teilmengen wiederum werden nach der gleichen Prozedur aufgeteilt. Diese Rekursion findet so lange statt, bis ein definiertes Endkriterium erfüllt ist und der Decision Tree zur Klassifikation bereit ist. Nachfolgend wird zuerst der Aufbau eines Decision Trees erklärt, sowie anschließend dessen Erstellung und Optimierung.

Anknüpfend an das das Beispiel der Kreditvergabe zeigt Abbildung 2.1 einen möglichen Decision Tree für diesen Anwendungsfall.

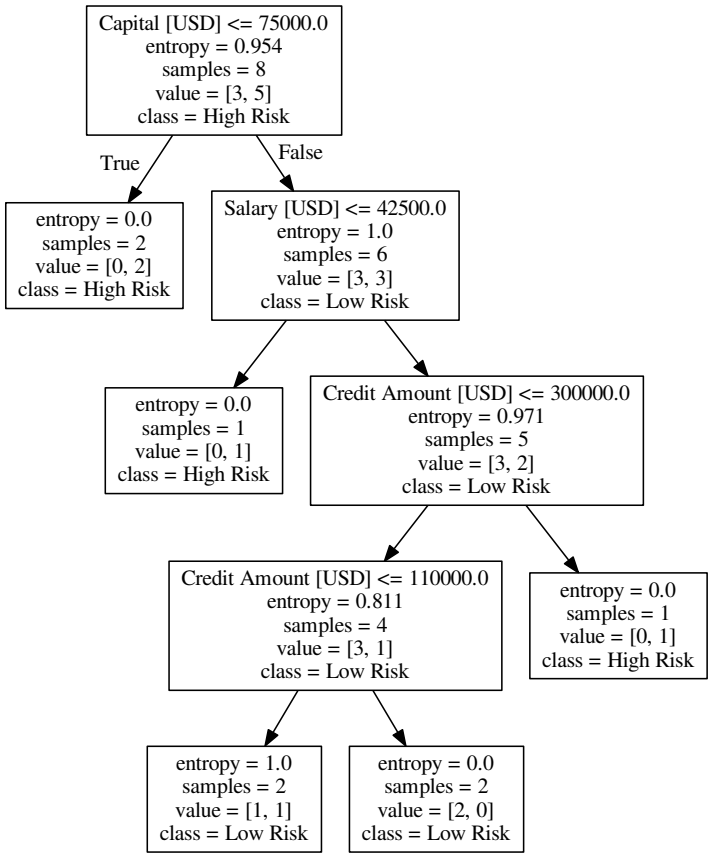

Abbildung 2.1 Beispielhafter Decision Tree für den Anwendungsfall der Kreditvergabe

Ein Decision Tree besteht aus der Wurzel, internen Knoten, Ästen und externen Knoten, genannt Blätter. Die Wurzel ist der einzige interne Knoten, der keinen Vorgänger hat. Ein interner Knoten ist ein Knoten, der Nachfolger hat. Interne Knoten testen jeweils ein bestimmtes Feature Φ der betrachteten Instanz X auf dessen Wert. Äste sind die Verbindungen zwischen Knoten. Jeder Ast bildet eine Wertemenge ab, die im vorgelagerten internen Knoten festgelegt wurde. Ein Blatt ist ein Knoten ohne Nachfolger. Blätter repräsentieren, im Fall der Klassifikation, die Klassen (Quinlan, 1986). In dieser Arbeit werden binäre Decision Trees

betrachtet, die genau zwei Klassen kennen. Auch finden ausschließlich univariate Bäume Anwendung, das heißt interne Knoten testen jeweils nur auf ein Feature, und nicht auf mehrere.

Die Klassifikation einer Instanz beginnt in der Wurzel des Decision Trees. Dort wird die Instanz auf ein Feature getestet und, je nach Wert dieses Features, entweder an den linken oder an den rechten Nachfolger der Wurzel weitergeleitet. Diese Prüfung mit Weiterleitung findet solange statt, bis die Instanz an einem Blatt angekommen ist. Dort wird der Instanz die Klasse des Blattes zugewiesen, wodurch diese Instanz klassifiziert ist (Russell u. Norvig, 2009). Der Graph aller Knoten, die die Instanz durchwandert hat, nennt sich Pfad.

Eine vorteilhafte Eigenschaft des Decision Trees ist die Best Case Laufzeit-Komplexität für die Klassifizierung von $\mathcal{O}(\log_2(n))$, die erreicht wird, wenn die Baumstruktur balanciert ist. Die Variable n steht für die Anzahl der Instanzen, mit denen der Decision Tree erstellt wurde, also $|\chi_{training}|$. Im Worst Case liegt die Komplexität bei $\mathcal{O}(n)$. Das ist der Fall, wenn für jedes ϕ_{opt} an jedem internen Knoten die Instanzen $X \in \chi_{training}$ so aufgeteilt werden, dass eine einzige Instanz als Blatt deklariert wird und die restlichen $n-1$ Instanzen in einen weiteren internen Knoten einfließen. Dort wiederholt sich der Vorgang rekursiv solange, bis alle Instanzen einem Blatt zugewiesen wurden und der Decision Tree fertig erstellt ist.

Ein weiterer Vorteil von Decision Trees ist, dass sie eine Menge von geordneten, simplen Regeln darstellen und somit für den Menschen leicht verständlich sind, wie die Regel aus dem Abschnitt 2.2 zeigt. So lässt sich zum Beispiel die Regel

$$\begin{aligned} &IF(Capital > 75.000 \wedge Salary \leqslant 42.500 \wedge CreditAmount > 300.000), \\ &\quad THEN\, Class = "HighRisk" \end{aligned} \tag{2.8}$$

aus der Abbildung 2.1 herleiten. Diese Verständlichkeit macht den Decision Tree zu einem sogenannten White Box Modell, dessen Entscheidungen für den Menschen nachvollziehbar sind. Black Box Modelle, wie der Random Forest oder neuronale Netze, hingegen treffen schwer- oder nicht-rekonstruierbare Entscheidungen (Gron, 2017). Wegen dieser Vorteile ist der Decision Tree Klassifikator sehr beliebt und wird in manchen Fällen den exakteren aber komplexeren Methoden vorgezogen (Alpaydın, 2008).

Im Gegensatz zu einigen statistischen Modellen wie der linearen oder exponentiellen Regression, gibt es im Funktionenraum F von Decision Trees stets eine Funktion $f(X|\Theta)$, die die Klassen aller Trainings-Instanzen korrekt abbildet. Ein Decision Tree kann prinzipiell beliebig viele Regeln definieren und somit jede Instanz aus $\chi_{training}$ korrekt abbilden. Diese Eigenschaft bringt allerdings einen Nachteil mit sich. Decision Trees sind für Overfitting (siehe Abschnitt 2.7) anfällig. Denn falls der Decision Tree, ohne Kontrolle der Baumstruktur, für jede Instanz einen eigenen Blattknoten anlegt, erzielt er zwar auf $\chi_{training}$ ein optimales Ergebnis, ist jedoch nicht zur Generalisierung und somit nicht zum Lernen fähig. Entsprechende Gegenmaßnahmen, um die Baumstruktur zu kontrollieren, finden sich im Unterabschnitt 2.3.2.

2.3.2 Erstellung eines Decision Trees

Die Induktion anhand des ID3-Algorithmus führt zu einem von mehreren äquivalenten Decision Trees. Es gibt mehr als einen Decision Tree, der die Regeln einer gegebenen Stichprobe $\chi_{training}$ kodiert (Quinlan, 1986). Nach Ockhams Rasiermesser ist es erstrebenswert, den Baum mit der geringsten Komplexität den anderen vorzuziehen (siehe Unterabschnitt 2.3.4). Diesen zu suchen ist jedoch ein NP-vollständiges Problem (Quinlan, 1986). Stattdessen bietet sich eine lokale Suche über Heuristiken an. Die nachfolgenden Lernalgorithmen sind vom Greedy-Typ: Von der Wurzel aus wählt der Algorithmus in jedem Schritt die in der aktuellen Position beste Aufteilung der Instanzen, um den Decision Tree iterativ zu erstellen (Alpaydın, 2008). Es folgt eine Beschreibung dieser Aufteilung sowie, anschließend, der Güte einer gegebenen Aufteilung.

Beginnend mit allen Trainingsinstanzen $\chi_{training}$ an der Wurzel, entsteht der Decision Tree durch rekursive Aufteilungen derselben an seinen internen Knoten. Eine Aufteilung bezeichnet hier das Bilden von Teilmengen der betrachteten Trainingsinstanzen an einem internen Knoten K_0, um pro Teilmenge einen Ast sowie einen weiteren Knoten K_1 zu generieren. Eine Aufteilung erfolgt stets anhand des optimalen Features ϕ_{opt}. Es sind zwei Fälle zu unterscheiden: entweder ist das Feature diskret oder es ist numerisch. Ist das Feature diskret, wird für jede Ausprägung dieses Features ein Ast generiert. Ist das Feature numerisch, wird es diskretisiert. Dazu werden ein oder mehrere Schwellenwerte festgelegt, die entsprechende Teilmengen definieren. Weist ein Feature ϕ_i zum Beispiel die Werte $[v_{start}, v_{end}]$ auf, könnte der Algorithmus die Schwellenwerte $\{s_1, s_2\}$ so wählen, dass gilt

$$v_{start} < s_1 < s_2 < v_{end}. \tag{2.9}$$

Die entstehenden Teilmengen wären dann

$$\begin{aligned} V_1 &= \{v | v < s_1\}, \\ V_2 &= \{v | v \geq s_1 \wedge v < s_2\} \text{ und} \\ V_3 &= \{v | v \geq s_2\} \end{aligned} \tag{2.10}$$

und würden zu drei entsprechenden Ästen führen. Der Spezialfall von genau einem Schwellenwert, wodurch sich zwei Teilmengen ergeben, wird als binäre Aufteilung bezeichnet (Alpaydın, 2008). Binäre Aufteilungen führen graphisch gesehen zu Hyperrechtecken, wie in Abbildung (TBD: DT Decision Boundaries Plot) dargestellt.

Die Anzahl der möglichen Aufteilungen ist gleich der Anzahl der vorhandenen Features, m. Die möglichen Aufteilungen unterscheiden sich hinsichtlich ihres Einflusses auf den finalen Decision Tree. Deshalb wird die Güte der möglichen Aufteilungen verglichen, um die sinnvollste Aufteilung zu identifizieren.

Als Kriterium für die Güte der Aufteilung verwendet der ID3-Algorithmus die aus ihr resultierende Änderung in Entropie, bezeichnet als Information Gain (IG) (Quinlan, 1986). Die Entropie H liegt per Definition zwischen Null und Eins und trifft eine Aussage über die Homogenität einer gegebenen Menge an n Instanzen. In der Informationstheorie bezeichnet die Entropie "die minimale Anzahl an Bits, die nötig sind, um die Klassifikationsgenauigkeit einer Instanz zu codieren"(Alpaydın, 2008). An jedem Knoten K wird die Entropie berechnet

mit (Shannon, 1948)

$$H(K) = -\sum_{w} p_i \log_2(p_i), \tag{2.11}$$

wobei w die möglichen Ausprägungen von Φ_{opt} bezeichnet. Die Variable p_i bezeichnet die relative Häufigkeit der Klasse y_i innerhalb der Instanzen, deren Ausprägung von Φ_{opt} gleich w ist. Besitzen alle Instanzen innerhalb der Ausprägungen w dieselbe Klasse, so ist die Entropie Null (da $\log_2(1) = 0$) und der Knoten bekommt keine Nachfolger sondern ist ein Blatt. Besitzt jedoch mindestens eine Instanz eine andere Klasse, so ist die Entropie größer als Null. Dann sucht ID3 nach jener Aufteilung, welche den größten IG mit sich bringt. Angenommen die Aufteilung erfolgt anhand von Feature ϕ_j. Dann ergibt sich der IG zwischen einem Knoten K_d der Tiefe d und seinen potenziellen Nachfolgern $K_d|\phi_j$ der Tiefe $d+1$ aus

$$IG(K_d, K_d|\phi_j) = H(K_d) - H(K_d|\phi_j). \tag{2.12}$$

Weiterhin angenommen, dass ϕ_j k Ausprägungen vorweist, dann folgt die Entropie der potenziellen Nachfolger der Tiefe $d+1$ aus

$$H(K_d|\phi_j) = \sum_{k} P(\phi_j = v_k) H(K_d|\phi_j = v_k), \tag{2.13}$$

wobei $P(\phi_j = v_k)$ die Wahrscheinlichkeit bezeichnet, dass ϕ_j den Wert v_k annimmt, und $H(K_d|\phi_j = v_k)$ die erwartete Entropie an jenem Knoten in Tiefe $d+1$ bezeichnet, an welchen eben diese Instanzen mit der Ausprägung v_k gelangen. Diese erwartete Entropie ergibt sich wiederum aus der Gleichung 2.11.

Da ID3 den IG maximiert, minimiert er die erwartete Anzahl an Schritten, die von der Wurzel bis zum Blatt nötig sind, also die Tiefe des Baumes. Eine minimale Tiefe wiederum bedeutet eine geringere Komplexität und ist nach Ockhams Rasiermesser den Alternativen vorzuziehen. Außerdem beschleunigt eine geringe Tiefe die Klassifikation, da die Instanz weniger Knoten und somit weniger Tests durchlaufen muss. Da der beschriebene Greedy-Algorithmus nur lokal sucht, ist das Ergebnis allerdings nicht zwangsläufig das globale Optimum.

Die Instanzen an allen internen Knoten in jeder Tiefe d werden nach dem beschriebenen Procedere aufgeteilt und den entstandenen Nachfolger-Knoten in Tiefe $d+1$ zugewiesen. Dort ruft sich der Algorithmus rekursiv auf. Sobald alle Instanzen an einem Knoten dieselbe Klasse aufweisen, wird dieser Knoten ein Blatt und die Instanzen werden nicht weiter aufgeteilt. Dem Blatt wird die Mehrheitsklasse seiner Instanzen zugewiesen. Pseudocode zur Erstellung eines Decision Trees ist in Algorithmus **??** zu finden. Die Laufzeitkomplexität zur Erstellung des Modells liegt bei $\mathcal{O}(m \times n \log_2(n))$, da an jedem Knoten alle m Features verglichen werden Gron (2017).

Russell u. Norvig (2009) weisen darauf hin, dass Features mit einem starken Verzweigungsfaktor nach dem ID3-Verfahren bevorzugt werden. Ein Beispiel dafür ist ein Zeitstempel, der für jede Instanz aus $\chi_{training}$ einzigartig ist. Dann würde eine Aufteilung anhand dieses Features stets zu einer erwarteten Entropie von Null führen, und würde somit stets den größtmöglichen IG erzielen. Der ID3-Algorithmus würde dieses Feature bevorzugen. Das würde zu starkem Overfitting führen, da das Modell nicht zur Generalisierung befähigt würde, also nicht lernt. Solchen stark-verzweigenden Features kann mit einer Bestrafung

basierend auf dem Verzweigungsfaktor, zum Beispiel mithilfe des Gain Ratios, entgegengewirkt werden (Russell u. Norvig, 2009). Ein Nachfolger des ID3-Algorithmus namens C4.5 berücksichtigt den Gain Ratio bei der Erstellung des Decision Trees (Gupta u. a., 2017).

Tabelle 2.1 (Hyper-)Parameter eines Decision Trees und deren Bedeutung

Name	Typ	Beschreibung	Zweck (Beispiele)
Maximale Anzahl an betrachteten Features	H	Beschränkung der Features, die pro Knoten verglichen werden	Beschleunigung des Trainings
Maximale Tiefe	H	Begrenzung der Tiefe des Decision Trees	Reduktion von Overfitting
Aufteilungskriterium	H	Kriterium zur Auswahl von Φ_{opt} für eine Aufteilung an Knoten K, z.B. IG oder Gini Index	Messung der Reinheit von Instanzen
Minimale Anzahl an Instanzen für Aufteilung	H	Schwellenwert bezüglich der Anzahl an Instanzen für eine weitere Aufteilung der Instanzen	Reduktion von Overfitting
Minimale Anzahl an Instanzen für Blatt	H	Schwellenwert bezüglich der Anzahl an Instanzen für die Erstellung eines neuen Blattes	Reduktion von Overfitting
Maximale Anzahl an Blättern	H	Begrenzung der Menge an Blättern im Decision Tree	Reduktion von Overfitting
Minimaler Unreinheits-reduktion für Aufteilung	H	Schwellenwert bezüglich des IG für eine weitere Aufteilung der Instanzen	Reduktion von Overfitting
Gewichtung der Instanzen in $\chi_{training}$	H	Gewichtung der Beispieldaten für das Training	Hervorhebung repräsentativer Instanzen bzw. Unterdrückung von Ausreißern
Tiefe	P	Tiefe des Decision Trees, Anzahl der Stufen von Wurzel bis zum entferntesten Blatt	Beurteilung der finalen Struktur
Anzahl an Blättern	P	Anzahl der Blätter im finalen Decision Tree	Beurteilung der finalen Struktur
Anzahl an internen Knoten	P	Anzahl an Knoten mit Nachfolgern (inklusive Wurzel, exklusive Blätter)	Beurteilung der finalen Struktur
Signifikanzen der Features	P	Bedeutung jedes Features Φ_i bei der Klassifizierung, z.B. berechnet anhand des IG durch Φ_i	Identifizierung der einfluss-reichsten Features
Anzahl der Features	P	Anzahl der Features, auf die die internen Knoten des Decision Trees testen	Beurteilung der finalen Struktur
Anzahl der Klassen	P	Anzahl der Klassen, die die Blätter des Decision Trees abbilden	Beurteilung der finalen Struktur

2.3.3 Grundlegende Parameter

Ein Machine Learning Algorithmus optimiert die Parameter Θ, um die Fehlerrate des Klassifikators zu minimieren (siehe Abschnitt 2.1). Die optimalen Werte, Θ_{opt}, resultieren also aus dem Trainingsprozess und sind erst im Nachhinein bekannt. Neben diesen Parametern gibt es Hyperparameter, die vom Anwender zu Beginn zu setzen sind. Einige dieser Hyperparameter beeinflussen die finalen Parameter Θ_{opt}, zum Beispiel indem sie den Wertebereich

einschränken. Die Tabelle 2.1 orientiert sich an der Implementierung von Scikit-learn (Pedregosa u. a., 2011) und gibt einen Überblick über grundlegende Parameter (Typ in der Tabelle ist "P") sowie Hyperparameter (Typ in der Tabelle ist "H") von Decision Trees und deren Bedeutung.

2.3.4 Optimierung von Decision Trees

Nach Erstellung ist ein Decision Tree oft zu komplex und neigt zu Overfitting (Gron, 2017). Eine Methode zur Reduzierung von Overfitting ist das Pruning, also das Kürzen des Baumes. Es wird zwischen Prepruning und Postpruning unterschieden. Prepruning findet noch während der Erstellung des Decision Trees statt. Ein Ansatz für Prepruning ist es, eine Bedingung für Aufteilungen an jedem Knoten zu definieren. Dadurch soll verhindert werden, dass Entscheidungen, die auf zu wenigen Instanzen basieren, die Varianz des Klassifikators erhöhen und somit die Generalisierung verschlechtern (Alpaydın, 2008). So könnte man festlegen, dass eine weitere Aufteilung nur dann stattfindet, wenn mindestens fünf Prozent der Trainingsinstanzen aus $\chi_{training}$ zu diesem Knoten gelangt sind. Andernfalls wird dieser Knoten zu einem Blatt, betitelt mit der Mehrheitsklasse.

Postpruning hingegen verändert den Baum, nachdem er komplett erstellt wurde. Die Decision Tree Pruning-Methode versucht jene Teilbäume zu identifizieren, welche für das Overfitting verantwortlich sind. Dazu wird vor dem Erstellen des Decision Trees eine Pruning-Menge $\chi_{pruning} \subsetneq \chi$ festgelegt, welche nicht in das Training einfließt. Das Training findet stattdessen mit den Instanzen aus $\chi \setminus \chi_{pruning}$ statt. Anschließend erfolgen pro Teilbaum zwei Messungen von Fehlerraten auf $\chi_{pruning}$. In der ersten Version klassifiziert der Decision Tree in seiner finalen Form die Instanzen aus $\chi_{pruning}$, ohne Veränderung des betrachteten Teilbaums. In der zweiten Version wird der jeweils betrachtete Teilbaum mit seiner Mehrheitsklasse ersetzt, also als Blatt simuliert. Ist der Fehler in der zweiten Version nicht signifikant schlechter als jener in der ersten Version, so wird der betrachtete Teilbaum dauerhaft in ein Blatt umgewandelt. Damit verringert sich die Komplexität und somit das Overfitting des Decision Trees (Russell u. Norvig, 2009). In der Praxis hat sich Postpruning als zielführender als Prepruning herausgestellt (Alpaydın, 2008).

2.4 Ensemble Methoden

2.4.1 Die Ensemble-Idee

Nach dem No-free-lunch Theorem ist kein Lernalgorithmus besser als ein anderer, sofern die durchschnittliche Fehlerrate über alle möglichen diskreten Funktionen betrachtet wird (Whitley u. Watson, 2005). Es gibt also keinen einzigen Lernalgorithmus, der in jedem Anwendungsfall das genaueste Modell erzeugt (Alpaydın, 2008). Ist der beste Klassifikator in einer gegebenen Situation gesucht, so wird dieser beispielsweise mit einem Validierungs-Datenset χ_{val} ermittelt. Eine Alternative, die die Auswahl eines einzigen besten Klassifikators umgeht, ist das Ensemble Learning. Die Idee dabei ist es, die – nicht zu 100% ausmerzbaren – Fehler an einer Instanz X_i von einzelnen Klassifikatoren durch andere Klassifikatoren, die X_i korrekt abbilden, überzukompensieren (Russell u. Norvig, 2009). Die einzelnen Basisklassifikatoren verhalten sich dann komplementär zueinander. Das Ziel ist es, dadurch die Fehlerrate insgesamt im Ensemble zu reduzieren.

Ein Ensemble ist ein Klassifikator, der aus mindestens zwei individuellen Klassifikatoren besteht. Die individuellen Klassifikatoren werden als Basisklassifikatoren bezeichnet. Als Basisklassifikatoren eigenen sich Modelle, die eine niedrigere Fehlerrate als zufälliges Raten erreichen. Ist dieses Kriterium erfüllt, so führt zum Beispiel das nachfolgend behandelte Boosting zu Ensembles mit "beliebig hoher Treffergenauigkeit"(Freund u. Schapire, 1997). Um eine möglichst große Fehlerreduktion im Vergleich zu den einzelnen Basisklassifikatoren zu erreichen, ist es nötig, diese Basisklassifikatoren maximal unterschiedlich zu machen, sie also zu dekorrelieren. Zur Messung der Dekorrelierung von Decision Trees schlägt Tin Kam Ho (1998) zum Beispiel die Metrik Tree Agreement vor. Für die Dekorrelierung gibt es verschiedene Ansätze. So können zum Beispiel verschiedene Lernalgorithmen genutzt werden, dieselben Lernalgorithmen mit unterschiedlichen Hyperparametern belegt werden oder die Basisklassifikatoren mit unterschiedlichen Repräsentationen der Input-Daten trainiert werden (Alpaydın, 2008). Eine weitere Möglichkeit zur Dekorrelierung der Basisklassifikatoren ist das Variieren der Trainingsdaten. Dazu gehören die Ansätze des Bagging und Boosting. Diese behandelt der nachfolgende Unterabschnitt 2.4.2.

2.4.2 Bagging und Boosting

Zwei verbreitete Methoden zur Dekorrelierung von Basisklassifikatoren in Ensembles sind das Bagging und das Boosting. Beide Methoden können sowohl auf Klassifikatoren als auch auf Regressions-Modelle angewendet werden (Alpaydın, 2008). Bagging steht für Bootstrap Aggregation. Nach dieser Methode werden für jeden Basisklassifikatoren mit dem Bootstrap-Algorithmus zufällige Instanzen aus χ ausgewählt. Das heißt, der jeweilige Basisklassifikatoren β_i wird mit der Stichprobe $\chi_i \subset \chi$ trainiert, wobei $|\chi_i| = |\chi|$. Diese Auswahl erfolgt mit Wiederholung, eine Instanz aus χ kann also mehrmals in χ_i vorkommen oder auch gar nicht. Die Wahrscheinlichkeit, dass eine Instanz X nicht in die Bootstrap-Stichprobe χ_i kommt ist Alpaydın (2008)

$$P(X \in \chi_i) = (1 - \frac{1}{|\chi|})^{|\chi|} \approx e^{-1} = 36,8\% \tag{2.14}$$

Das heißt im Umkehrschluss, dass χ_i ungefähr $100\% - 36,8\% = 63,2\%$ der Instanzen aus χ enthält. Im Gegensatz zum Boosting entstehen die Stichproben χ_i beim Bagging unabhängig voneinander.

Boosting bedeutet Verstärken. Dabei sind die Stichproben nicht unabhängig voneinander, sondern werden der Reihe nach gebildet. Die Idee von Boosting ist es, die Stichprobe χ_{i+1} dahingehend anzupassen, dass sie jene Instanzen aus χ_i bevorzugt, welche von dem darauf trainierten Basisklassifikatoren nicht erfolgreich gelernt wurden. Der Basisklassifikatoren β_{i+1} soll also von den Fehlern seines Vorgängers β_i lernen. Somit sollen die Basisklassifikatoren ihre Schwächen untereinander ausgleichen. Um den Bedarf nach einer sehr großen Datenmenge zum Boosting zu beseitigen, entwarfen Freund u. Schapire (1997) die Variante AdaBoost, oder ausgeschrieben Adaptive Boosting. Außerdem ist AdaBoost im Gegensatz zu seinen Vorgängern dazu in der Lage, beliebig viele Basisklassifikatoren zu kombinieren (Alpaydın, 2008).

2.4.3 Entscheidungsfindung im Ensemble

Die Entscheidung, welche Klasse einer neuen Instanz X zugewiesen wird, trifft ein Ensemble auf Basis der Entscheidungen seiner Basisklassifikatoren. Ein Ensemble E, bestehend aus den n Basisklassifikatoren $\{\beta_1, ..., \beta_n\}$ unter Verwendung von Kombinationsmethode Ψ, kann als Vorhersagefunktion für Instanz X als

$$E(X|\{\beta_1, ..., \beta_n\}, \Psi) = \hat{y} \tag{2.15}$$

formalisiert werden. Für Ψ lassen sich einstufige und mehrstufige Kombinationsmethoden unterscheiden (Alpaydın, 2008). Diese Arbeit beschränkt sich auf einstufige Methoden. Eine einstufige Methode ist die nachfolgend beschriebene Voting-Methode.

Wenn $\{\hat{y}_1, ..., \hat{y}_n\}$ die Ergebnisse von $\{\beta_1, ..., \beta_n\}$ sind, dann ergibt sich nach der Voting-Methode Ψ_{voting} die Entscheidung des Ensembles $\hat{y}_E$ allgemein (im Sinne von sowohl für Klassifikation als auch für Regression gültig) mit

$$\hat{y}_E = \Psi_{voting}(\hat{y}_1, ..., \hat{y}_n). \tag{2.16}$$

Handelt es sich bei $\{\hat{y}_1, ..., \hat{y}_n\}$ um die vorhergesagten Klassen, so nennt man dies Hard Voting. Handelt es sich dabei jedoch um die Wahrscheinlichkeiten für die positive Klasse, so spricht man von Soft Voting Gron (2017). Im Spezialfall, dass das Ensemble als Klassifikator verwendet wird, entspricht das finale Ergebnis der nächstgelegenen – gemessen an der Distanz zu $\hat{y}_E$ – Klasse. Das heißt in einer binären Klassifikation mit den Klassen $y \in \{0, 1\}$ ergibt sich das Ergebnis $\hat{y}_{Class}$ mit

$$\hat{y}_{Class} = \begin{cases} 1(\text{positiv}), & \text{wenn } 0 \leqslant \hat{y}_E < 0,5 \\ 0(\text{negativ}), & \text{wenn } 0,5 \leqslant \hat{y}_E \leqslant 1. \end{cases} \tag{2.17}$$

Die Annahme bezüglich Ψ_{voting} ist, dass alle Basisklassifikatoren $\{\beta_1, ..., \beta_n\}$ gleich gewichtet sind, und zwar mit einem Gewicht von $\frac{1}{n}$. In der Literatur finden sich Vorschläge, wie man die Basisklassifikatoren gewichten kann. Ein solches Vorgehen ist es, die Basisklassifikatoren auf einem Validierungs-Datenset χ_{val} zu bewerten, um sie im Ensemble dann anhand ihrer Treffgenauigkeit zu gewichten (Alpaydın, 2008).

An dieser Stelle sei, alternativ zur Voting-Methode, die geschachtelte Generalisierung von Wolpert (1992) erwähnt. Dessen Idee ist es, die Kombination von $\hat{y}_1, ..., \hat{y}_n$ nicht durch eine (gewichtete) Summe zu vollziehen, sondern selbst wiederum durch eine lernende Funktion Ψ_{sg} zu entwerfen. Das finale Ergebnis eines Ensembles, das eine solche Kombinationsfunktion verwendet, lautet dann

$$\hat{y}_E = \Psi_{sg}(\hat{y}_1, ..., \hat{y}_n|\Theta). \tag{2.18}$$

Ein Lernalgorithmus sucht die optimalen Parameter Θ_{opt}, sodass die Fehlerfunktion minimiert wird, wie in Abschnitt 2.1 beschrieben. Die Kombinationsfunktion Ψ_{sg} soll also die Schwächen der einzelnen Basisklassifikatoren lernen, um diese bei der finalen Kombination zu berücksichtigen (Alpaydın, 2008). Da nun sowohl die Base Classifer, die $\{\hat{y}_1, ..., \hat{y}_n\}$ erzeugen, als auch die Kombinationsfunktion Ψ_{sg} selbst anhand von Daten lernen, findet eine geschachtelte Generalisierung statt.

2.5 Random Forest Klassifikator

2.5.1 Generelle Funktionsweise und Eigenschaften

Ein Random Forest ist ein Ensemble, das Decision Trees als Basisklassifikatoren nutzt. Trotz ihrer vergleichsweise simplen Struktur gehören Random Forests zu den mächtigsten Machine Learning Algorithmen (Gron, 2017). Durch das Einführen von Zufälligkeit ist ein Random Forest robuster als ein einzelner Decision Tree. Die Zufälligkeit entsteht durch Bagging und durch zufällige Auswahl jener Attribute, die für einen Split überhaupt erst betrachtet werden. Diese Zufälligkeit hilft dabei, unkorrelierte Decision Trees zu generieren, die ihre Schwächen gegenseitig ausgleichen und somit komplementär zueinander sind.

Die Klassifikation durch einen Random Forest erfolgt nach der Voting-Methode (siehe Unterabschnitt 2.4.3). Die Decision Trees aus dem Random Forest klassifizieren die Instanz nach dem in Abschnitt 2.3.1 dargelegten Procedere. Die Ergebnisse fließen als gleich-gewichtete Stimmen in den Random Forest ein. Die am häufigsten vorkommende Klasse ist das Ergebnis der Klassifizierung.

Der Funktionenraum F eines Random Forests enthält – wie auch der von Decision Trees – stets eine Funktion f, die die Klassen aller Trainings-Instanzen korrekt abbildet. Der entscheidende Unterschied zu Decision Trees jedoch ist die Lösung des Overfitting-Problems. Mithilfe des Gesetzes der großen Zahlen hat Breiman (2001) bewiesen, dass bei Random Forests kein Overfitting auftritt. Der Generalisierungsfehler eines Random Forests hängt von der Güte der einzelnen Decision Trees ab als auch von der Korrelation dergleichen ab (Breiman, 2001).

Eine vorteilhafte Eigenschaft ergibt sich aus der Tatsache, dass der Random Forest Bagging verwendet. Wie in Kapitel 2.4.2 gezeigt, enthalten die Stichproben, auf denen die Basisklassifikatoren trainiert werden, durchschnittlich $63,2\%$ der Instanzen aus χ. Die restlichen $36,8\%$ der Instanzen aus χ werden als out-of-bag (OOB) bezeichnet. Diese OOB Instanzen können zur Erfolgsmessung verwendet werden, noch bevor ein Validierungs- oder Test-Datenset zum Einsatz kommt. Die gemessene Fehlerrate nennt sich in dem Fall OOB Error (Gron, 2017).

2.5.2 Erstellung eines Random Forests

Ein Random Forest entsteht durch das Trainieren von n Decision Trees auf den Trainingssets $\{\chi_1, \chi_n\}$, welche mittels Bagging (siehe Kapitel 2.4.2) aus χ generiert werden.

Eine Besonderheit dabei ist, dass nicht alle Features für die Aufteilung an Knoten K betrachtet werden. Stattdessen erwägt der Random Forest Algorithmus eine zufällige (engl. random), echte Untermenge der Features. Von den i Attributen der Daten wird eine Untermenge von j Attributen zufällig ausgewählt. Für diese j Attribute werden im nächsten Schritt jeweils die Information Gains berechnet, die aus ihren Aufteilungen (siehe Unterabschnitt 2.3.2) resultieren würden. Das Attribut mit dem höchsten Information Gain wird für den betrachteten Knoten als Aufteilungskriterium gewählt. Diese Zufälligkeit dekorreliert die Basisklassifikatoren, denn diese treffen ihre Entscheidungen basierend auf verschiedenen Features, und ermöglicht dadurch ein komplementäres Ensemble. Algorithmus 1 veran-

Algorithm 1 Random Forest Pseudocode

Require: Trainingset $\chi_{training}$, Features Φ, Anzahl der Decision Trees n

function RandomForest $(\chi_{random\_forest}\Phi_{random\_forest}, n)$
$T \leftarrow \emptyset$
for $i \in \{1, ..., n\}$ **do**
  $\chi_i \leftarrow$ Bootstrapping-Set von χ_{random_forest}
  $\Phi_i \leftarrow$ Randomisierte, echte Teilmenge von Φ_{random_forest}
  $t_i \leftarrow$ DecisionTree(χ_i, Φ_i)
  $T \cup t_i$
end for
return T
end function

function DecisionTree$(\chi_{decision\_tree}\Phi_{decision\_tree})$
$K \leftarrow$ Wurzel-Knoten k_1
while K enthält mindestens einen internen Knoten k_{intern_neu} ohne Nachfolger **do**
  $\Phi_{opt} \leftarrow$ Feature mit höchstem Information Gain an Knoten k_{intern}
  Teile Instanzen an k_{intern_neu} entsprechend der Werte von Φ_{opt} auf neue Knoten K_{neu} auf
  Weise k_{intern_neu} alle Knoten aus K_{neu} als Nachfolger zu
  for $K_{neu\_i} \in K_{neu}$ **do**
    if Instanzen an Knoten K_{neu_i} haben alle dieselbe Klasse **then**
      Erkläre K_{neu_i} als Blatt mit Mehrheitsklasse
    else
      Erkläre K_{neu_i} als internen Knoten ohne Nachfolger
    end if
  end for
  $K \cup K_{neu}$
end while
return K
end function

schaulicht die Erstellung eines Random Forests mittels Pseudocode.

2.5.3 Grundlegende (Hyper-)Parameter

Zusätzlich zu den (Hyper-)Parametern seiner Basisklassifikatoren (siehe 2.3.3) besitzt der Random Forest als Ensemble noch weitere Einstellungen. Die Tabelle 2.2 orientiert sich an der Implementierung von Scikit-learn (Pedregosa u. a., 2011) und gibt einen Überblick über alle relevanten (Hyper-)Parameter.

Aufbauend auf der ursprünglichen Idee von Breiman (2001) entwickelten Forscher neue Random Forest Modelle mit dem Ziel, die Fehlerrate weiter zu reduzieren. Ebenso optimieren manche Ansätze den Random Forest für bestimmte Anwendungsbereiche, oder reduzieren die Trainings- und Klassifikationszeit. Diese Arbeit stellt einige dieser weitergehenden Random Forest Variationen im Unterabschnitt **??** vor.

Tabelle 2.2 (Hyper-)Parameter eines Random Forests und deren Bedeutung

Name	Typ	Beschreibung	Zweck (Beispiele)
Anzahl der Basis-klassifikatoren	H	Anzahl der Decision Trees innerhalb des Random Forests	Steigerung der Treffer-genauigkeit durch mehr komplementäre Basisklassifikatoren
Bootstrap-Sampling	H	Bestimmung, ob Basis-klassifikatoren auf den gesamten Daten oder auf Bootstrap-Stichproben trainiert werden	Steigerung der Zufälligkeit bei Training der Basis-klassifikatoren durch Veränderung der Trainingsstichproben
OOB Treffer-genauigkeit	P	Schätzung der Treffergenauig-keit, gemessen den den out-of-bag Instanzen der jeweiligen Decision Trees	Beurteilung der Treffer-genauigkeit noch vor Evaluierung auf einem Validierungs- oder Testdatenset
Alle (Hyper-) Parameter von Decision Trees	H/P	Die (Hyper-)Parameter der Basisklassifikatoren werden über den Random Forest zentral definiert	Steigerung der Treffer-rate des Random Forests durch Optimierung der Hyperparameter seiner Decision Trees

2.6 Erfolgsmessung von Klassifikatoren

Nach der Erstellung eines oder mehrerer Klassifikatoren ist häufig die Güte des Modells von Interesse, beispielsweise gemessen an der Fehlerrate. Fehlerraten werden unter anderem verglichen, um den geeignetsten Klassifikator für einen gegebenen Anwendungsfall zu bestimmen. Außerdem ist die erwartete Fehlerrate auf neuen Daten außerhalb $\chi_{training}$ häufig von Interesse. Das ist insbesondere der Fall, wenn es für die Fehlerrate eine – selbstauferlegte oder fremdbestimmte – harte Obergrenze von p% gibt, die nachgewiesen werden muss, bevor der Klassifikator in der Realität angewendet werden darf.

Das Training, die Validierung und das Testen erfolgen jeweils auf den separaten Datensets $\chi_{training}$, $\chi_{validierung}$ und χ_{test}, wobei gilt

$$\chi_{training} \cap \chi_{validierung} = \emptyset; \chi_{validierung} \cap \chi_{test} = \emptyset \text{ und } \chi_{training} \cap \chi_{test} = \emptyset. \tag{2.19}$$

$\chi_{training}$ dient der Optimierung der Parameter eines Klassifikators, $\chi_{validierung}$ dem Tuning von Hyperparametern und χ_{test} der Erhebung des erwarteten Fehlers auf neuen Daten. Jedes dieser Datensets soll die Grundgesamtheit, aus der die Stichproben stammen, repräsentativ darstellen. Alpaydın (2008) schlägt vor, ein Drittel der vorhandenen Daten für χ_{test} zu verwenden und die anderen zwei Drittel auf $\chi_{training}$ und $\chi_{validierung}$ zu verteilen. Für diese Einteilung zwischen $\chi_{training}$ und $\chi_{validierung}$ gibt es mehrere Varianten. Eine davon ist die Kreuzvalidierung.

In der Kreuzvalidierung werden aus χ k Stichproben, $\{\chi\_1, ..., \chi_k\}$ mit $|\chi\_1| = ... = |\chi\_k|$ generiert. Dann erfolgen k Durchläufe, genannt Folds, wobei der Klassifikator in $Fold_i$ auf

$\chi_1 \cup ... \cup \chi_{i-1} \cup \chi i + 1 \cup ... \cup \chi_k$ trainiert wird und auf χ_i getestet wird. Alpaydın (2008) merkt an, dass für jede Teilstichprobe χ_i die relative Häufigkeit jeder Klasse y gleich sein sollte, und, dass diese wiederum den relativen Häufigkeiten von y in χ gleichen sollten. Dieser Vorgang nennt sich Stratifizierung. Ein Spezialfall der Kreuzvalidierung ist die Leave-One-Out Methode Alpaydın (2008). Dabei wird χ in $k = n$ Stichproben aufgeteilt, mit $n = |\chi_{training} \cup \chi_{validierung}|$. Jede Trainingsstichprobe χ_i besteht also aus einer einzigen Instanz. In diesem Spezialfall ist Stratifizierung nicht möglich. Die Leave-One-Out Methode wird zum Beispiel in der medizinischen Diagnostik verwendet, wenn nur sehr wenige relevante Daten vorhanden sind Alpaydın (2008). Eine weitere Variante zur Einteilung der Datensets $\chi_{training}$, $\chi_{validierung}$ und χ_{test} ist die Bootstrap Methode. Diese wurde bereits in Kapitel 2.4.2 behandelt.

Um die Güte eines Klassifikators zu bestimmen und mit anderen zu vergleichen, bieten sich verschiedene Kennzahlen an. Jede Klassifizierung fällt in eines der vier Felder aus der Konfusionsmatrix in Tabelle 2.3.

Tabelle 2.3 Konfusionsmatrix

		$\hat{y}$ (Ergebnis der Klassifikation)	
		1	0
y	1	WP (Wahre Positive)	FN (Falsche Negative)
	0	FP (Falsche Positive)	WN (Wahre Negative)

Die Fehlerrate E berechnet sich dann mit

$$E = \frac{|FP| + |FN|}{N}, \tag{2.20}$$

wobei $N = |WP| + |FP| + |FN| + |WN|$.

Im Falle von verzerrten Datensets, wenn die Klassen sehr ungleich verteilt sind, eignet sich diese simple Fehlerrate alleine nicht zur Erfolgsmessung; stattdessen bieten sich weitergehende Metriken aus der Konfusionsmatrix an (Gron, 2017). Precision misst den Erfolg des Klassifikators auf den positiven Instanzen und ergibt sich aus

$$Precision = WP/(WP + FP). \tag{2.21}$$

Precision alleine reicht jedoch nicht aus. Ein Klassifikator könnte von n wahren Positiven nur einen einzigen, sicheren als positiv bestimmen, wobei $1 << n$. Damit wäre die Precision $\frac{1}{(1+0)} = 100\%$, jedoch zu Lasten von $n - 1$ falschen Negativen. Um diesen Trade-Off zu erkennen, bietet sich der Recall als weitere Metrik an. Dieser folgt aus

$$Recall = WP/(WP + FN). \tag{2.22}$$

Das F-Maß kombiniert schließlich Precision und Recall als harmonischer Durchschnitt (Gron, 2017)

$$F - Ma = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = 2x\frac{PrecisionxRecall}{Precision + Recall}. \tag{2.23}$$

Das F-Maß bestraft zwar Klassifikatoren, die sehr ungleiche Werte für Precision und Recall erzielen. Gleichzeitig bevorzugt es jedoch Klassifikatoren, die für Precision und Recall ähnliche Werte aufweisen. Das ist nicht immer erstrebenswert. Je nach Anwendungsfall kann entweder Precision oder Recall relevanter sein (Russell u. Norvig, 2009). Handelt es sich um die Klassifikation zur Bombendetektion, so würde man vermutlich Fehlalarme einer entgangenen Bombe vorziehen. Dann wäre Recall wichtiger als Precision. Allgemein geht eine höhere Precision mit niedrigerem Recall einher, und vice versa (Gron, 2017). Dieser Precision-Recall Trade-Off ist eine zentrale Fragestellung im Machine Learning und ist für jeden Anwendungsfall einzeln zu lösen.

Es sei erwähnt, dass die Konfusionsmatrix nicht das einzige Kriterium zur Erfolgsmessung eines Klassifikators sein sollte. **?** ergänzt zum Beispiel die Speicher- und Laufzeit-Komplexität sowie die Interpretierbarkeit als weitere Bewertungskriterien, die in der Forschung noch nicht genug Beachtung gefunden hätten.

2.7 Underfitting, Overfitting und der Curse of Dimensionality

Um die bestmögliche Generalisierung zu erreichen, vergleicht man die Komplexität der erlernten Funktion f mit der den Daten zugrundeliegenden Funktion (Alpaydın, 2008). Ist die Komplexität von f niedriger als die der approximierten Funktion, wird dies als Underfitting bezeichnet. Dies ist zum Beispiel der Fall, wenn man versucht, eine Gerade auf einen Datensatz anzupassen, der von einem Polynom dritten Grades stammt (Alpaydın, 2008). Die Erhöhung der Komplexität führt bei Underfitting zu einer Verbesserung der Vorhersagegenauigkeit beziehungsweise zu einer Reduktion des Vorhersagefehlers. Ist die Komplexität von f höher als die der approximierten Funktion, so wird dies als Overfitting bezeichnet. Ein Beispiel ist der oben erwähnte, nicht-kontrollierte Decision Tree, der für jede Instanz einen eigenen Blattknoten anlegt. Es folgt die Definition von Overfitting. Dabei bezeichnet $E(.)$ die Fehlerfunktion. Angenommen, der Lernalgorithmus zieht die Funktion $f_1(X)$ aus F einer Funktion $f_2(X)$ vor, weil auf $\chi_{training}$ gilt:

$$E(f_1(x)) < E(f_2(X)). \tag{2.24}$$

Gleichzeitig aber gilt auf der gesamten Verteilung, aus der $\chi_{training}$ stammt:

$$E(f_1(X)) > E(f_2(X)). \tag{2.25}$$

Dann wird die Funktion $f_1(X)$ als overfitted bezeichnet (**?**).

Was genau die Komplexität ausmacht, hängt vom betrachteten Klassifikator ab. In Decision Trees leitet sich die Komplexität aus der Anzahl an Knoten und Blättern ab. Eine große Anzahl an Knoten und Blättern bedeutet hohe Komplexität, was wiederum zu Overfitting führen kann, und vice versa. Für Decision Tree Ensembles, wie dem Random Forest, kommt zusätzlich noch die Anzahl der einzelnen Basisklassifikatoren als Faktor für die Komplexität hinzu. Hierbei ist anzumerken, dass eine größere Anzahl an Basisklassifikatoren aber nicht unbedingt zu Overfitting führt. Random Forests profitieren – im Widerspruch zu Ockhams Rasiermesser – von einer steigenden Anzahl an Basisklassifikatoren, solange diese hinreichend unabhängig voneinander sind (Gron, 2017).

Eine weitere, allgemeine Herausforderung – unabhängig vom konkreten Anwendungsfall – im Machine Learning ist der Curse of Dimensionality. Dieses Phänomen bezeichnet die exponentielle Steigung der benötigten Features, die bei Erhöhung der Dimensionen nötig ist (Verleysen u. François, 2005). Wenn beispielsweise ein Klassifikator in einer Dimension mit 10 Instanzen trainiert wird, so sind in zwei Dimensionen 100 und in drei Dimensionen 1.000 Trainings-Instanzen nötig, um den gleichen Lernerfolg zu erzielen (Verleysen u. François, 2005). Obwohl Random Forests dazu in der Lage sind, den Curse of Dimensionality zu reduzieren, sind diese dennoch von einer steigenden Anzahl an Features negativ betroffen, da sich die Trainingszeit verlängert (Li, 2016). Li (2016) schlug einen mittels Hadoop MapReduce parallelisierten Random Forest vor und zeigte, dass dieser verglichen mit einem nicht-parallelisierten Random Forest ein besseres Ergebnis sowie eine um 40% verkürzte Rechenzeit erreicht.

2.8 Besonderheiten bei der Klassifikation von Zeitreihen

Für den Spezialfall der Zeitreihen-Klassifikation bieten sich angepasste Methoden für die Messung der Fehlerrate an. Es ist zum Beispiel möglich, dass zwischen den Instanzen aus χ zeitliche Abhängigkeiten bestehen. So könnte in einer Stichprobe über die Jahre j_1 bis j_n ein Muster ab Jahr j_s auftreten, wobei $j_1 < j_s < j_n$. Ein Beispiel ist die Einführung eines neuen Gesetzes in Jahr j_s, das die betrachtete Zeitreihe ab dann beeinflusst. In diesem Fall sollte der Klassifikator das Muster nur auf Instanzen aus den Jahren $]j_s, j_n]$, nicht jedoch aus der Vorzeit $[j_1, j_s]$ anwenden.

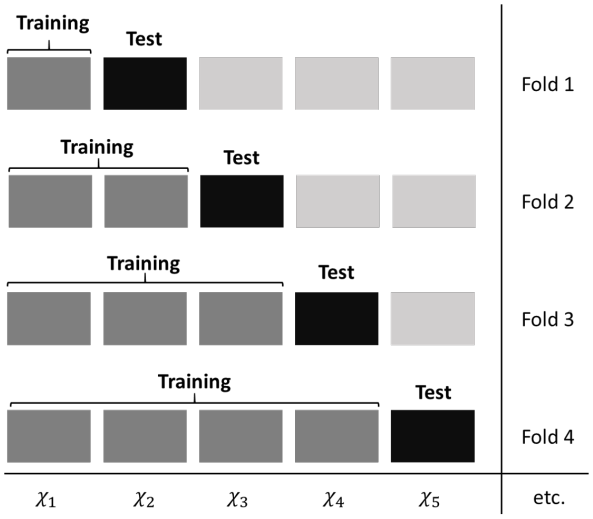

Abbildung 2.2 Schematischer Ablauf einer Time Series Cross-Validation in den ersten vier Iterationen

Ein Ansatz, um zeitlich-bedingte Muster zu entdecken, ist die Blocked Form Cross-Validation, auch bekannt als Time Series Cross-Validation. Bergmeir u. Benitez (2011) haben empirisch ermittelt, dass diese Methode für Zeitreihen-Klassifikatoren genauere Ergebnisse erzielt als herkömmliche Cross-Validation Varianten wie die erwähnte Kreuzvalidierung. Der schematische Ablauf ist in Abbildung 2.2 dargestellt. Ausgehend von der Trainings-Stichprobe $\chi_{training}$ bildet die Time Series Cross-Validation k Teilstichproben, $\{\chi_1, ..., \chi_k\}$,

wobei $|\chi\_1| = ... = |\chi\_k|$. Dann erfolgen $i = k - 1$ Folds, wobei der Klassifikator in $Fold_i$ jeweils auf $\chi_1 \cup ... \cup \chi_i$ trainiert wird und auf χ_{i+1} getestet wird. Alternativ kann $\chi_i \cup ... \cup \chi_k$ als Testset verwendet werden. Eine weitere Variation ist es, nicht auf den Daten aller vorangegangener Zeitabschnitte, sondern lediglich auf jenen des letzten Zeitabschnittes, also auf χ_i, zu trainieren (Cerqueira u. a., 2019).

2.9 Random Walk Theorie und Markteffizienzhypothese

Da sich diese Arbeit mit Anwendungsfällen aus der Finanzdomäne beschäftigt, folgen zwei grundlegende Theorien aus diesem Gebiet. Nach der Random Walk Theorie (**?**) sind sämtliche Analysen zur Vorhersage von Aktienkursen wertlos. Das gilt sowohl für die technische Analyse, die annimmt, dass es in Aktienkursen sich wiederholende Muster gibt, als auch für die Fundamentalanalyse, die den intrinsischen Wert eines Unternehmens anhand von Finanzkennzahlen bewertet. Die technische Analyse bezieht sich ausschließlich auf die Zeitreihen eines Wertpapiers. Kennzahlen über die prozentuale oder absolute Veränderung des Kurses sowie statistische Werte bilden dabei die Grundlage für Kauf- und Verkaufsentscheidungen. Ein Beispiel aus der technischen Analyse ist der gleitende Durchschnitt der letzten d Tage, wobei d beliebig variiert wird. Die Fundamentalanalyse hingegen basiert auf den Finanzkennzahlen des Emittenten des Wertpapiers, und nicht auf dem Kurs des Wertpapiers. Beispiele für solche Kennzahlen sind der Umsatz, der Jahresüberschuss oder auch die Eigenkapitalrendite des Unternehmens. Ebenso betrachtet die Fundamentalanalyse ökonomische Faktoren wie den Leitzins, gesellschaftliche Trends oder das Management des Emittenten.

Analysen basieren stets auf Informationen, die den Marktteilnehmern bekannt sind. Nach der Markteffizienzhypothese sind all diese Informationen jedoch bereits im Preis des Wertpapiers abgebildet. Dann kann die Analyse der Informationen keine überproportionalen Renditen ermöglichen. Nach **?** ist ein effizienter Markt ein Markt, in dem sehr viele rationale, Gewinn-maximierende Akteure konkurrieren, wobei jeder von diesen Zugriff auf alle Informationen hat und versucht, die Kursbewegungen gewinnbringend vorherzusagen. Somit spiegelt der Preis eines Wertpapiers in einem effizienten Markt alle vorhandenen Informationen wieder. Sowohl Ereignisse aus der Vergangenheit als auch zukünftig erwartete Ereignisse, über die sich die Marktakteure einig sind, bestimmen die Preisbildung. Der Marktpreis stimmt also mit dem inneren Wert des Wertpapiers überein. In einem solchen Markt "hat eine Zeitreihe von Aktienkursen kein Gedächtnis"(**?**); es ist nicht möglich von historischen Beobachtungen auf neue Kursbewegungen zu schließen. Erkennbare, gewinnbringende Muster würden von der Masse der rationalen Akteure sofort ausgenutzt, bis sie im Marktpreis berücksichtigt und somit nutzlos wären (**?**). Also können nur neue Informationen zu einer Änderung des inneren Preises eines Wertpapiers führen. Und neue Informationen können per Definition nicht vorhergesagt werden (**?**).

In der Realität treffen einige Annahmen der Markteffizienzhypothese allerdings nicht immer zu. Zum einen handeln die Akteure am Aktienmarkt, also Käufer und Verkäufer, nicht immer rational. Auch wenn Computer einen wachsenden Teil der Aktienkäufe und -Verkäufe tätigen, sind emotional geleitete Menschen am Markt aktiv. Auch hat nicht jeder Marktteilnehmer die gleichen Informationen. Zum Beispiel nutzen die Akteure unter-

schiedliche Datenquellen, die Informationen verzerren oder zu verschiedenen Zeitpunkten veröffentlichen. Selbst wenn die Informationsbasis aller Akteure gleich wäre, würde jeder von diesen – aufgrund einmaliger Erfahrungen und Fähigkeiten eines jeden Menschen – zu unterschiedlichen Ergebnissen kommen.

Weiterhin ist es möglich, dass die handelnden Computerprogramme selbst, durch ihre automatischen Kauf- und Verkaufsaktionen bei Unter- oder Überschreitung gewisser Preispunkte, Muster in den Aktienkursen erzeugen. Dann könnte ein Klassifikator, wie der Decision Tree, eben diese Muster erkennen und Regeln bilden, um sie systematisch auszunutzen.

Diese über fünfzig Jahre alten Theorien wurden in jüngerer Zeit kritisch hinterfragt. Vor dem Hintergrund steigender Rechenleistung von Computern und gehäufter Erfolge im Machine Learning hat man mit verschiedenen Klassifikatoren versucht, signifikant höhere Renditen zu erzielen als der Marktdurchschnitt und die Random Walk Theorie zu widerlegen. Ausgewählte Arbeiten zu diesem Thema finden sich in Unterabschnitt **??**.

2.10 Next: Aktuelle Literatur & Aufzeigen von Lücken, auf denen Methodik aufbaut

3 Ergebnisse

3.0.1 Durchstich 1: DT Hyperparameter MAX_DEPTH

Wie auf Abbildung 3.1 dargestellt, erzielt der Decision Tree Klassifikator in diesem Beispielfall für MAX_DEPTH = 9 das beste Ergebnis.

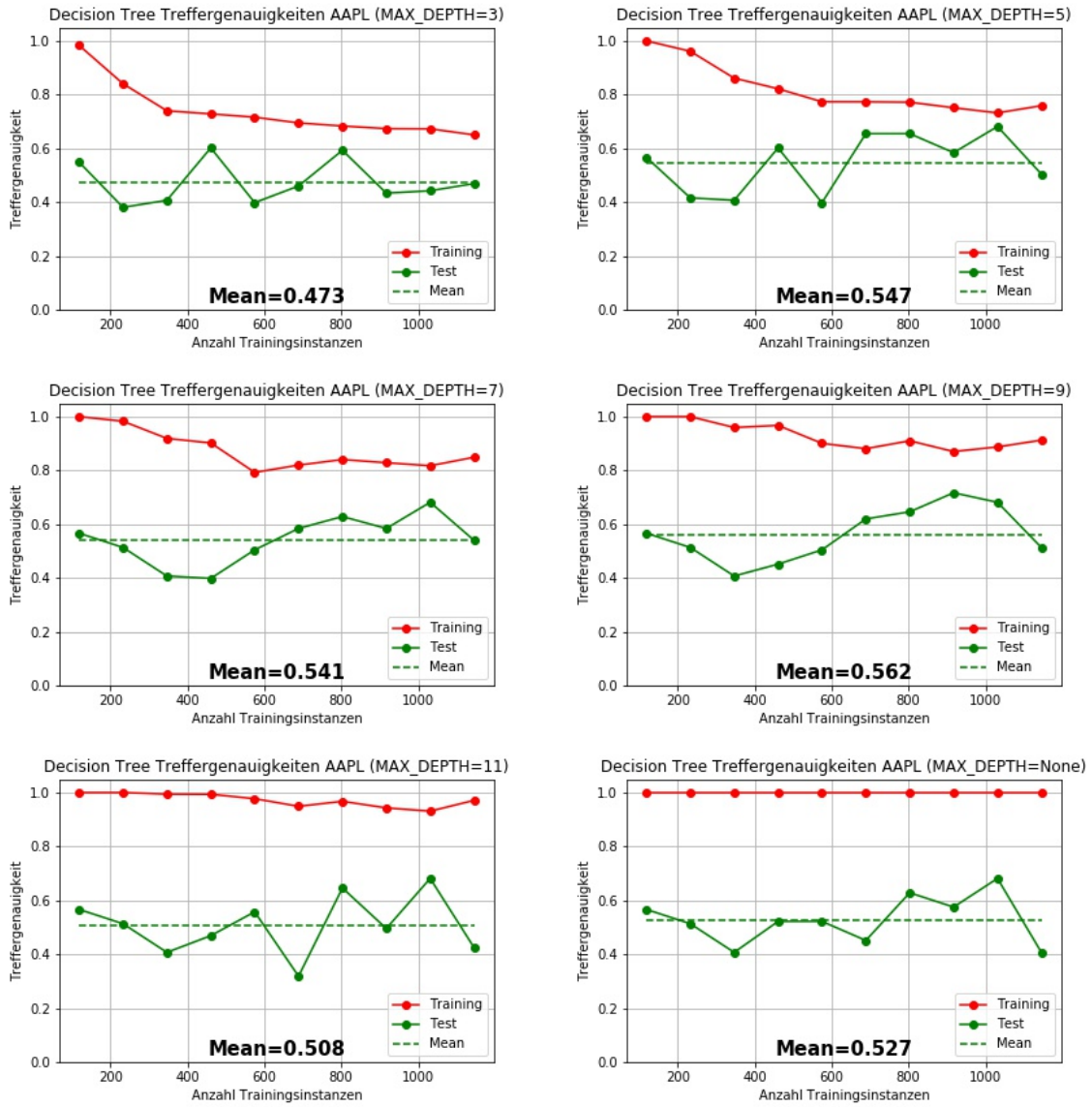

Abbildung 3.1 Treffgenauigkeiten (vertikale Achse) eines Decision Trees für verschiedene MAX_DEPTH Hyperparameter (3, 5, 7, 9, 11, None) mit zunehmendem Trainingsset (horizontale Achse)

Mit steigender MAX_DEPTH ist eine kontinuierliche Zunahme der Treffgenauigkeit auf dem Trainigsset zu beobachten. Die Treffgenauigkeit auf dem Testset jedoch nimmt nur bis MAX_DEPTH=9 trendmäßig zu. Anschließend fällt sie ab, da das Modell zu komplex wird und die Generalisierungsfähigkeit sinkt. Das Overfitting Problem tritt also in diesem Szenario ab MAX_DEPTH = 9 auf. Es ist anzumerken, dass sich die optimalen Parameter aus dem Hyperparameter Tuning jedoch von diesem Wert unterscheiden können. Die GridSearch variiert nämlich neben MAX_DEPTH weitere Hyperparameter des Decision Trees, wie zum Beispiel MIN_SAMPLES_LEAF, und kann Overfitting somit auf anderem Wege reduzieren.

Literaturverzeichnis

[Abdou u. Pointon 2011] Abdou, Hussein ; Pointon, John: Credit Scoring, Statistical Techniques and Evaluation Criteria: A Review of the Literature. In: *Int. Syst. in Accounting, Finance and Management* 18 (2011), 04, S. 59–88. http://dx.doi.org/10.1002/isaf.325. – DOI 10.1002/isaf.325

[Alpaydın 2008] Alpaydin, E.: *Maschinelles Lernen*. Oldenbourg, 2008. – ISBN 9783486581140

[Bergmeir u. Benitez 2011] Bergmeir, C. ; Benitez, J. M.: Forecaster performance evaluation with cross-validation and variants. In: *2011 11th International Conference on Intelligent Systems Design and Applications*, 2011, S. 849–854

[Breiman 2001] Breiman, Leo: Random Forests. In: *Machine Learning* 45 (2001), Oktober, Nr. 1, S. 5–32. http://dx.doi.org/10.1023/A:1010933404324. – DOI 10.1023/A:1010933404324. – ISSN 0885–6125

[Cerqueira u. a. 2019] Cerqueira, Vitor ; Torgo, Luis ; Mozetic, Igor: *Evaluating time series forecasting models: An empirical study on performance estimation methods*. 2019

[Cutler u. a. 2007] Cutler, D. R. ; Edwards, Thomas C. ; Beard, Karen H. ; Cutler, Adele ; Hess, Kyle ; Gibson, Jacob ; Lawler, Joshua J.: Random forests for classification in ecology. In: *Ecology* 88 11 (2007), S. 2783–92

[Diaz-Uriarte u. Alvarez 2006] Diaz-Uriarte, Ramon ; Alvarez, Sara: Gene Selection and Classification of Microarray Data Using Random Forest. In: *BMC bioinformatics* 7 (2006), 02, S. 3. http://dx.doi.org/10.1186/1471-2105-7-3. – DOI 10.1186/1471–2105–7–3

[Freund u. Schapire 1997] Freund, Yoav ; Schapire, Robert E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In: *Journal of Computer and System Sciences* 55 (1997), Nr. 1, S. 119 – 139. http://dx.doi.org/https://doi.org/10.1006/jcss.1997.1504. – DOI https://doi.org/10.1006/jcss.1997.1504. – ISSN 0022–0000

[Gron 2017] Gron, Aurlien: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017. – ISBN 1491962291, 9781491962299

[Gupta u. a. 2017] Gupta, Bhumika ; Rawat, Aditya ; Jain, Akshay ; Arora, Arpit ; Dhami, Naresh: Analysis of Various Decision Tree Algorithms for Classification in Data Mining. In: *International Journal of Computer Applications* 163 (2017), 04, S. 15–19. http://dx.doi.org/10.5120/ijca2017913660. – DOI 10.5120/ijca2017913660

[Howard u. Bowles 2012] Howard, Jeremy ; Bowles, Mike: *The Two Most Important Algorithms in Predictive Modeling Today*. https://conferences.oreilly.com/strata/strata2012/public/schedule/detail/22658, 2012. – Besucht: 19. Oktober 2019

[Li 2016] Li, C.: The application of high-dimensional data classification by random forest based on hadoop cloud computing platform. 51 (2016), 01, S. 385–390. http://dx.doi.org/10.3303/CET1651065. – DOI 10.3303/CET1651065

[Pedregosa u. a. 2011] Pedregosa, Fabian ; Varoquaux, Gaël ; Gramfort, Alexandre ; Michel, Vincent ; Thirion, Bertrand ; Grisel, Olivier ; Blondel, Mathieu ; Prettenhofer, Peter ; Weiss, Ron ; Dubourg, Vincent ; Vanderplas, Jake ; Passos, Alexandre ; Cournapeau, David ; Brucher, Matthieu ; Perrot, Matthieu ; Duchesnay, Édouard: Scikit-learn: Machine Learning in Python. In: *J. Mach. Learn. Res.* 12 (2011), November, S. 2825–2830. – ISSN 1532–4435

[Prasad u. a. 2006] Prasad, Anantha ; Iverson, Louis ; Liaw, Andy: Newer Classification and Regression Tree Techniques: Bagging and Random Forests for Ecological Prediction. In: *Ecosystems* 9 (2006), 03, S. 181–199. http://dx.doi.org/10.1007/s10021-005-0054-1. – DOI 10.1007/s10021–005–0054–1

[Quinlan 1986] Quinlan, J. R.: Induction of decision trees. In: *Machine Learning* 1 (1986), Mar, Nr. 1, S. 81–106. http://dx.doi.org/10.1007/BF00116251. – DOI 10.1007/BF00116251

[Russell u. Norvig 2009] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA : Prentice Hall Press, 2009. – ISBN 0136042597, 9780136042594

[Shannon 1948] Shannon, Claude E.: A Mathematical Theory of Communication. In: *The Bell System Technical Journal* 27 (1948), 7, Nr. 3, S. 379–423. http://dx.doi.org/10.1002/j.1538-7305.1948.tb01338.x. – DOI 10.1002/j.1538–7305.1948.tb01338.x

[Shotton u. a. 2011] Shotton, J. ; Fitzgibbon, A. ; Cook, M. ; Sharp, T. ; Finocchio, M. ; Moore, R. ; Kipman, A. ; Blake, A.: Real-time Human Pose Recognition in Parts from Single Depth Images. In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, 2011 (CVPR '11). – ISBN 978–1–4577–0394–2, S. 1297–1304

[Svetnik u. a. 2003] Svetnik, Vladimir ; Liaw, Andy ; Tong, Christopher ; Culberson, J. C. ; Sheridan, Robert P. ; Feuston, Bradley P.: Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. In: *Journal of Chemical Information and Computer Sciences* 43 (2003), Nr. 6, S. 1947–1958. http://dx.doi.org/10.1021/ci034160g. – DOI 10.1021/ci034160g. – PMID: 14632445

[Tang u. a. 2018] Tang, Cheng ; Garreau, Damien ; Luxburg, Ulrike von: When do random forests fail?, 2018

[Tin Kam Ho 1998] Tin Kam Ho: The random subspace method for constructing decision forests. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20 (1998), Aug, Nr. 8, S. 832–844. http://dx.doi.org/10.1109/34.709601. – DOI 10.1109/34.709601

[Verleysen u. François 2005] Verleysen, Michel ; François, Damien: The Curse of Dimensionality in Data Mining and Time Series Prediction, 2005, S. 758–770

[Whitley u. Watson 2005] Whitley, L. D. ; Watson, Jean-Paul: Complexity Theory and the No Free Lunch Theorem, 2005

[Wolpert 1992] Wolpert, David H.: Stacked generalization. In: *Neural Networks* 5 (1992), Nr. 2, S. 241 – 259. http://dx.doi.org/https://doi.org/10.1016/S0893-6080(05)80023-1. – DOI https://doi.org/10.1016/S0893–6080(05)80023–1. – ISSN 0893–6080