Hochschule Karlsruhe
University of
Applied Sciences

-Machine Learning: Boosting-Algorithmen-

Seminararbeit

Student:	David Erdös	67906
Universität:	Hochschule Karlsruhe – Technik und Wirtschaft	
Studiengang:	Informatik Bachelor	
Semester:	Wintersemester 2023	
Dozent:	Prof. Dr. Baier	
Bearbeitet am:	26. November 2023	

Inhaltsverzeichnis

1 Einleitung

In der dynamischen Welt des maschinellen Lernens (ML) haben sich Boosting-Algorithmen als revolutionär erwiesen, indem sie beeindruckende Leistungen in einer Vielzahl von Anwendungen erbringen [1, Kapitel 1.1.2]. Besonders bei tabellarischen Datensätzen erzielen sie in fast allen Bereichen die besten Ergebnisse.
In dieser Seminararbeit liegt der Schwerpunkt auf AdaBoost und GradientBoosting, die gemäß Géron [2, S. 192] zu den bekanntesten Vertretern der Boosting-Familie gehören. Die Arbeit wird die beiden Algorithmen detailliert untersuchen und verständlich darstellen.

1.1 Motivation und Zielsetzung

Das Ziel dieser Seminararbeit ist es, ein tiefes Verständnis für die Funktionsweise von AdaBoost und GradientBoosting zu entwickeln und deren Einsatzmöglichkeiten anhand konkreter Beispiele zu demonstrieren. Hierbei wird besonderer Wert darauf gelegt, zunächst die Funktionsweise verständlich zu erläutern.

1.2 Struktur der Arbeit

Die Seminararbeit gliedert sich in mehrere Abschnitte, die schrittweise aufeinander aufbauen. Zunächst wird ein Überblick über die Grundlagen des maschinellen Lernens gegeben, gefolgt von einer detaillierten Betrachtung der Boosting-Algorithmen, insbesondere AdaBoost und GradientBoosting. Anschließend erfolgt ein Vergleich dieser beiden Methoden, wobei deren Stärken und Schwächen in verschiedenen Anwendungsszenarien beleuchtet werden. Abschließend wird ein Ausblick auf zukünftige Entwicklungen und mögliche Forschungsrichtungen im Bereich des Boostings gegeben.

[3, text] [1, text] [2, text] [4, text] [5, text]

2 Grundlagen des Machine Learning (3-4 Seiten)

2.1 Modern Approaches in Machine Learning

Überblick über aktuelle Trends und Innovationen im Machine Learning Vorstellung fortgeschrittener Techniken und Methoden Diskussion über die Bedeutung von Deep Learning und künstlichen neuronalen Netzen

2.2 Role of Boosting Algorithms in ML

Einführung in Boosting-Algorithmen und ihre Relevanz Spezifische Betrachtung von AdaBoost und Gradient Boosting Vergleich von Boosting-Algorithmen mit anderen fortgeschrittenen Methoden

2.3 Boosting Algorithms in Tabular Data Analysis

Bedeutung von tabellenartigen Datensätzen in fortgeschrittenen ML-Anwendungen Analyse, wie AdaBoost und Gradient Boosting bei tabellenartigen Daten effektiv sind Fallstudien und Beispiele aus der Praxis, die den Einsatz dieser Algorithmen zeigen

Daten Entscheidungsbäume Traingsfehler

3 Boosting

Nach meiner eigenen Erfahrung als Student fällt oft auf, dass Gruppenarbeit nicht immer die erwarteten Vorteile bringt. Oft dominiert in Lerngruppen ein einzelner Student, der über fundiertes Wissen verfügt, die Diskussion und die Gruppenleistung spiegelt im Wesentlichen seine individuelle Leistung wider.
In Fällen, wo alle Mitglieder einer Lerngruppe nur begrenztes Wissen zu einem Thema haben, kann die kollektive Leistung sogar hinter dem zurückbleiben, was man durch zufällige Antworten erwarten würde. Die Vorstellung, dass eine Gruppe von Individuen mit begrenztem Wissen gemeinsam Ergebnisse erzielen kann, die sowohl den Durchschnitt als auch jede individuelle Bestleistung übertreffen, widerspricht oft unserer Intuition. Selbst historische Weisheiten, wie sie in der Bibel gefunden werden, betonen die Risiken einer solchen Zusammenarbeit. Dort heißt es metaphorisch: „Wenn aber ein Blinder den andern führt, so fallen sie beide in die Grube“ (Mt 23,16; Mt 23,24; Lk 6,39; Röm 2,19).

Doch im Bereich des maschinellen Lernens offenbart sich ein ganz anderes Szenario. Hier ermöglicht das Boosting-Verfahren, dass die Kombination von schwachen Modellen zu einem leistungsstarken Gesamtsystem führt. Dieser Ansatz, der die aggregierte Intelligenz mehrerer einfacher Modelle nutzt, um komplexe Probleme zu lösen, steht im starken Gegensatz zu den oft enttäuschenden Ergebnissen menschlicher Gruppenarbeit mit begrenztem Wissen [1, S. 3].

3.1 Was ist Boosting am Beispiel Wettererkennung

Boosting (ursprünglich Hypothesis Boosting) bezeichnet eine beliebige Ensemble-Methode, bei der sich mehrere schwache Lerner zu einem starken Lerner kombinieren lassen Géron [2, S. 191].

Die genannte Definition des Boosting lässt sich gut anhand des Beispiels der Wettererkennung veranschaulichen. Betrachten wir folgende einfache Regeln (Schwache Lerner) zur Beurteilung, ob es regnet:

Schwache Lerner	**Grenzwert**
Nasser Boden	Ja
Wolken am Himmel	Ja
Hohe Luftfeuchtigkeit	$> 80\%$
Personen mit Regenschirm	Ja
Außentemperatur	$> 0°C$

Tabelle 1: Individuelle Vorhersagen der Schwache Lerner

Beispielsweise ist ein nasser Boden zwar eine Voraussetzung und ein guter erster Filter, allerdings könnte der Boden genauso gut durch einen Rasensprenger nass sein.
Die Temperatur ist hingegen ein relativ schlechtes Indiz für die Frage, ob es gerade regnet. Es unterscheidet aber den Fall Regen und Schnee und ist somit trotzdem essentiell für die Klassifikation.

3.1.1 Erzeugung eines starken Lerners

Der nächste Schritt ist es, die Aussagen der schwachen Lerner in ein nützliches Model zusammenzufassen, einen sog. 'starken Lerner'. Im einfachsten Fall lässt man die schwachen Lerner mit gleicher Stimmkraft abstimmen. Aus dem Stimmverhältnis der Vorhersagen der schwachen Lerner lässt sich in unserem Fall die Regenwahrscheinlichkeit ableiten.

Nasser Boden	Wolken	Hohe Luftfeuchte	Regenschirm	Über 0°C	Regenwahrscheinlichkeit (%)
Ja	Ja	Ja	Ja	Ja	100%
Ja	Ja	Nein	Nein	Ja	60%
Nein	Ja	Ja	Ja	Ja	80%
Ja	Nein	Ja	Nein	Ja	60%
Nein	Nein	Nein	Nein	Ja	20%
Ja	Ja	Ja	Nein	Nein	60%
Nein	Ja	Nein	Ja	Ja	80%
...	...	...	...	...	...

Tabelle 2: Wahrheitstabelle zur Vorhersage von Regen basierend auf schwachen Lernern. Die Werte dienen in erster Linie der Veranschaulichung.

3.2 Gewichtete Abstimmung

Aus der Mehrheitsentscheidung, bei der alle schwachen Lerner die gleiche Stimmkraft haben, eine Vorhersage zu treffen, liefert nur selten die besten Ergebnisse. Eine bessere Vorgehensweise, die auch von vielen Boosting-Algorithmen benutzt wird, ist die gewichtete Abstimmung. Diese Methode berechnet die Stimmkraft oder Gewichtung des Lerners α individuell, wie im Algorithmus 1.1 von Schapire und Freund beschrieben [1, S. 5].

3.2.1 Grundlegende Annahmen

Gegeben:
- Ein Datensatz X, bestehend aus n Beispielpaaren, wobei jedes Paar aus einem Eingabedatum x_i und dem entsprechenden Zielwert y_i besteht: $X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.
- Schwache Lerner t mit Vorhersagefunktion/Hypthese h_t.

Jeder schwache Lerner trifft Vorhersagen basierend auf dem Datensatz X. Der Erfolg dieser Vorhersagen wird durch den Fehler ε_t gemessen.

3.2.2 Fehlerberechnung

Der Fehler ε_t eines schwachen Lerners ist der Anteil der falschen Vorhersagen:

$$\varepsilon_t = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(h_t(x_i) \neq y_i), \tag{1}$$

wobei $\mathbb{I}$ die Indikatorfunktion ist, die 1 ist, wenn $h_t(x_i) \neq y_i$ (d.h., die Vorhersage ist falsch), und 0 sonst.

3.2.3 Gewichtung der schwachen Lerner

Die Gewichtung α_t jedes schwachen Lerners basiert auf seinem Fehler ε_t:

$$\alpha_t = \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right). \tag{2}$$

Ein zuverlässiger Lerner t hat eine höhere Gewichtung α_t je nach dem wie groß sein Fehler ε_t ist.

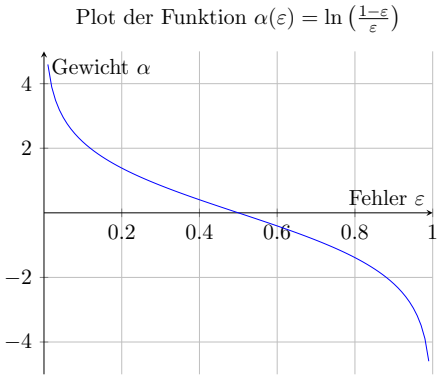

Abbildung 1: Visualisierung der Gewichtsfunktion α in Abhängigkeit vom Fehler ε

3.2.4 Endgültige Vorhersage des starken Lerners

Die Gesamtvorhersage H des Boosting-Modells ergibt sich aus der gewichteten Abstimmung aller schwachen Lerner.

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right). \tag{3}$$

In anderen Worten wird die Vorhersage eines Lerners auf den Daten $h_t(x)$ mit seiner Gewichtung α multipliziert. Die Summe dieses Produkts für alle Lerner T ergibt die Gesamtvorhersage H. Hier wird lediglich noch die Signum-Funktion angewendet, um den Wert zwischen 0 und 1 zu glätten.

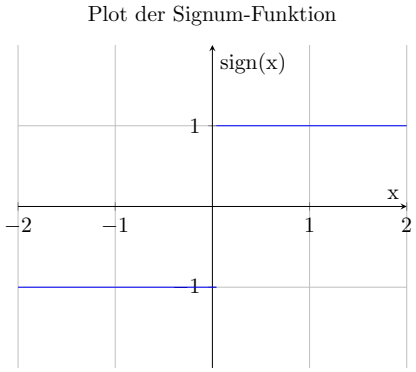

Abbildung 2: Visualisierung der Signum-Funktion

3.3 Annahme des schwache Lernens

Die Voraussetzung jedes Boosting-Algorithmus ist ein bereits vorhandener schwacher Basis-Lernalgorithmus. Das Ziel ist es dann, durch mehrfache Ausführung des Boosting-Algorithmus' die Leistung des Lernalgorithmus zu verbessern, bzw. zu 'boosten'. Laut Schapire und Freund [1, S. S. 4] reicht es aus geringe Ansprüche an die Leistung der schwachen Lerner zu stellen. Es ist völlig hinreichend, wenn der Lernalgorithmus Hypothesen liefert mit etwas besseren Ergebnissen als 50% Fehlerquote, was dem uninformierten Raten gleich käme.
Diese Annahme, dass der Lerner schwache Hypothesen hervorbringt, die mindestens etwas besser sind als zufälliges Raten, wird die *Annahme des schwachen Lernens genannt.*

4 AdaBoost

4.1 Adaptives Lernen

Ein Großteil der Boosting-Algorithmen benutzt das Prinzip des sequenziellen adaptiven Lernens. Das heißt, dass man die Lerner sequentiell, also nacheinander, so trainiert, dass jeder Lerner versucht adaptiv die Fehler des vorherigen Lerners auszubessern.
Der Vertreter dieser Methode ist der AdaBoost-Algorithmus, dessen Name sich von 'Adaptive Boosting' ableitet.

4.1.1 Gewichtete Daten

Um einen Lerner in einem Boosting-Algorithmus zu entwickeln, wird der Lernalgorithmus zunächst auf dem ursprünglichen Datensatz angewendet. Nach dieser ersten Runde bewertet der Algorithmus die Klassifikationsergebnisse, um festzustellen, welche Datenpunkte richtig und welche falsch klassifiziert wurden. Die falsch klassifizierten Datenpunkte erhalten dann ein höheres Gewicht, was ihre Bedeutung im Datensatz für den nächsten Schritt erhöht. Im Gegenzug werden die Gewichte der korrekt klassifizierten Datenpunkte verringert.
In der darauffolgenden Iteration wird der Lernalgorithmus erneut angewendet, diesmal aber auf den Datensatz, dessen Gewichte gerade angepasst wurden. Dadurch wird der Lernalgorithmus versuchen sich besonders auf die vom ersten Lerner falsch klassifizierten Daten zu konzentrieren. Nach dem Training des zweiten Lerners erfolgt eine weitere Anpassung der Gewichte nach den gleichen Regeln.
Dieser Prozess wird fortgesetzt, wobei jeder nachfolgende Lerner, darauf abzielt, die Fehler seiner Vorgänger zu korrigieren (siehe Abbildung 3).

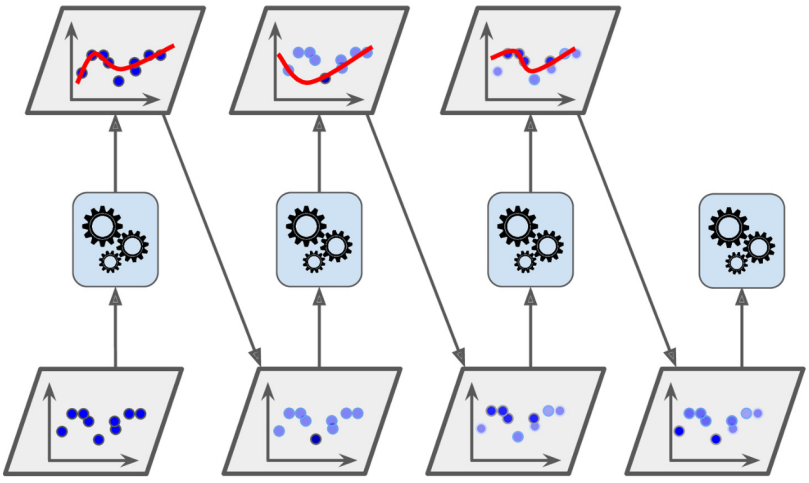

Abbildung 3: Sequentielles Training mit aktualisierten Gewichten der Datenpunkte bei AdaBoost (Quelle: Géron [Abbildung 7-7 2, S. 192])

4.2 Algorithmus-Struktur und Funktionsweise

Der AdaBoost-Algorithmus (siehe Algorithmus 1) ist in mehrere Schritte unterteilt, welche im folgenden genau erklärt werden:

Schritt 1: Initialisierung (Zeile 2) Hier werden die Gewichte $D_1(i)$ initial gleich verteilt. Das heißt, dass jeder Datenpunkt am Anfang das gleiche Gewicht hat, also gleich wichtig ist.

Schritt 2: Iterationen (Zeile 3 bis 13) Der Algorithmus durchläuft T Iterationen, in denen pro Durchlauf ein neuer schwacher Lerner trainiert und bewertet wird.

Schritt 2.1: Training der schwachen Lerner (Zeile 4-5) In jeder Iteration t wird ein schwacher Lerner h_t unter der aktuellen Gewichtungsverteilung D_t der Daten trainiert. Dieser Lerner erzeugt eine Hypothese zur Klassifizierung der Daten.

Schritt 2.2: Fehlerberechnung (Zeile 6) Der Fehler ε_t des Lerners wird berechnet als Anteil der falsch klassifizierten Beispiele, gewichtet nach D_t. Weitere Details zur Fehlerberechnung sind in Unterunterabschnitt 3.2.2 erläutert.

Schritt 2.3: Gewichtung der Lerner (Zeile 7) Die Gewichtung α_t jedes schwachen Lerners basiert auf seinem Fehler ε_t und wird mithilfe der Formel $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$ berechnet. Diese Gewichtung bestimmt den Einfluss jedes Lerners auf das Endergebnis. Genaueres dazu ist im Unterunterabschnitt 3.2.3 zu finden.

Schritt 2.4: Aktualisierung der Gewichte (Zeile 8-12) Die Gewichte der Datenpunkte $D_t(i)$ werden für die nächste Iteration $D_{t+1}(i)$ aktualisiert. Datenpunkte, die falsch klassifiziert wurden, erhalten ein höheres Gewicht, während korrekt klassifizierte Datenpunkte ein niedrigeres Gewicht bekommen. Die Aktualisierung erfolgt gemäß der Formel in Zeile 9. Z_t ist dabei ein Normalisierungsfaktor, der sicherstellt, dass D_{t+1} eine Wahrscheinlichkeitsverteilung bleibt.

Schritt 3: Ausgabe der finalen Hypothese (Zeile 14) Nach Abschluss aller Iterationen wird die finale Hypothese $H(x)$ als gewichtete Summe der Hypothesen aller schwachen Lerner ausgegeben:

$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Diese Vorhersagefunktion repräsentiert das Endergebnis des AdaBoost-Algorithmus.

Algorithm 1 AdaBoost Algorithmus (nach Schapire und Freund, Algorithmus 1.1, S. 5)

Gegeben: Trainingssatz $(x_1, y_1), \ldots, (x_m, y_m)$, wobei $x_i \in X$, $y_i \in \{-1, +1\}$
Initialisierung: $D_1(i) = \frac{1}{m}$ für $i = 1, \ldots, m$
for $t = 1, \ldots, T$ **do**
    Trainiere schwachen Lerner mit Verteilung D_t
    Erhalte Hypothese $h_t : X \to \{-1, +1\}$
    Berechne Fehler $\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$
    Wähle $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    Aktualisiere Gewichte:
    for $i = 1, \ldots, m$ **do**

$$D_{t+1}(i) = \frac{D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$$

    end for
    wobei Z_t ein Normalisierungsfaktor ist (gewählt so dass D_{t+1} eine Verteilung ist)
end for
Ausgabe: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

4.3 Anwendungsbeispiel

Um den doch eher abstrakten AdaBoost-Algorithmus 1 zu veranschaulichen, wird im folgenden die Funktionsweise anhand des Beispiels in Abbildung 4 beschrieben. Die Werte der Berechnung sind in Tabelle 3 zu finden.

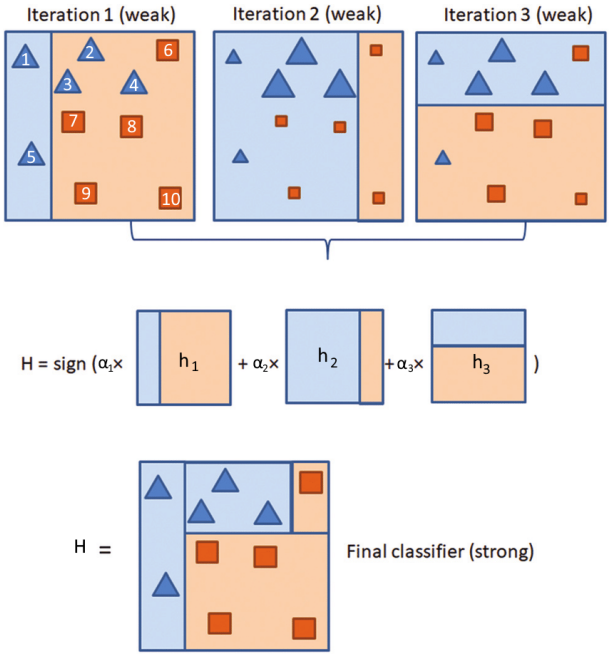

Abbildung 4: Schematische Darstellung des AdaBoost-Prozesses (Quelle: ResearchGate, Alex Henderson et al.)

.

Im Trainingssatz gibt es 10 Datenpunkte i, die entweder durch ein blaues Dreieck oder ein orangenes Viereck dargestellt sind. Wie gefordert, sind zu Beginn der 1. Iteration alle Datenpunkte mit den gleichen Gewichten D_1 initialisiert, was durch die gleiche Größe der geometrischen Formen veranschaulicht wird.

Unter der Annahme des schwachen Lernens aus Unterabschnitt 3.3 nehmen wir an, dass der genutzte schwache Lernalgorithmus die Fläche entweder horizontal, oder vertikal mit einer Linie teilt. Die Hypothese h_1 der 1. Iteration identifiziert 7 von 10 Datenpunkten richtig und hat somit eine Fehlerquote ϵ_1 von 0.30. Tatsächlich gibt es keine Linie, die alle Formen richtig zuweisen kann, 7 von 10 ist also ein passables Ergebnis.

Die aus der Fehlerquote ϵ_1 berechnete Gewichtung α_1 beträgt 0.42. Es lässt sich erkennen, dass die falsch klassifizierten Dreiecke 2-4 eine Erhöhung ihrer Gewichte mit dem Faktor $e^{-\alpha_1 y_i h_1(x_i)}$, der ungefähr 1.53 beträgt, erfahren, während die richtig klassifizierten Formen durch einen Faktor von etwa 0.65 eine Verringerung ihrer Gewichte erfahren.
Die aktualisierten Gewichte $D_2(i)$ für die zweite Iteration ergeben sich als Produkt der vorherigen Gewichte $D_1(i)$ und des Faktors $e^{-\alpha_1 y_i h_1(x_i)}$. Diese neuen Gewichte werden dann durch den Normalisierungsfaktor $Z_1 \approx 0.92$ geteilt, um sicherzustellen, dass die Gesamtsumme aller D_t gleich 1 bleibt.

Diese Abfolge an Schritten aus Algorithmus 1 wiederholt sich in jeder Iteration. In diesem Beispiel gibt es insgesamt drei Iterationen. Nach der 3. Iteration werden die Hyptohesen h mit ihrer Gewichtung α in der finalen Hypthese H zusammengefasst.
Wie aus Abbildung 4 hervorgeht, reichen drei Iterationen aus, um einen starken Lerner hervorzubringen, welches alle Datenpunkte i richtig klassifiziert. Es ist unerwartet, dass die Fehlerrate auf den Trainingsdaten so rapide abnimmt. Der exponentielle Abstieg der Fehlerquote ist tatsächlich typisch für AdaBoost. Dies ist eine der Eigenschaften die viele Boosting-Algorithmen so gebräuchlich machen.

Tabelle 3: Berechnungen für Abbildung 4 gemäß Algorithmus 1 nach Vorlage von Schapire und Freund [Tabelle 1.1 1, S. 9]

	1	2	3	4	5	6	7	8	9	10	
$D_1(i)$	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	$\epsilon_1 = 0.30$
$e^{-\alpha_1 y_i h_1(x_i)}$	0.65	1.53	1.53	1.53	0.65	0.65	0.65	0.65	0.65	0.65	$\alpha_1 \approx 0.42$
$D_1(i) \cdot e^{-\alpha_1 y_i h_1(x_i)}$	0.07	0.15	0.15	0.15	0.07	0.07	0.07	0.07	0.07	0.07	$Z_1 \approx 0.92$
$D_2(i)$	0.07	0.17	0.17	0.17	0.07	0.07	0.07	0.07	0.07	0.07	$\epsilon_2 = 0.21$
$e^{-\alpha_2 y_i h_2(x_i)}$	0.52	0.52	0.52	0.52	0.52	0.52	1.91	1.91	1.91	0.52	$\alpha_2 \approx 0.65$
$D_2(i) \cdot e^{-\alpha_2 y_i h_2(x_i)}$	0.04	0.09	0.09	0.09	0.04	0.04	0.14	0.14	0.14	0.04	$Z_2 \approx 0.82$
$D_3(i)$	0.05	0.11	0.11	0.11	0.05	0.05	0.17	0.17	0.17	0.05	$\epsilon_3 = 0.10$
$e^{-\alpha_2 y_i h_2(x_i)}$	0.33	0.33	0.33	0.33	3.00	3.00	0.33	0.33	0.33	0.33	$\alpha_3 \approx 1.10$
$D_3(i) \cdot e^{-\alpha_3 y_i h_3(x_i)}$	0.02	0.04	0.04	0.04	0.15	0.15	0.06	0.06	0.06	0.02	$Z_3 \approx 0.64$

5 Gradient Boosting

Der Gradient Boosting Trees (GBT) Algorithmus hat sich in den letzten Jahren als einer der führenden Boosting-Algorithmen etabliert. Charakteristisch für GBT ist die Verwendung von Entscheidungsbäumen als Basis-Lernalgorithmus. Ähnlich wie AdaBoost verfolgt GBT das Prinzip des adaptiven Lernens (Unterabschnitt 4.1), bei dem jeder nachfolgende Entscheidungsbaum darauf abzielt, die Fehler seines Vorgängers zu korrigieren.

5.1 Unterschied zu AdaBoost

Der entscheidende Unterschied zu AdaBoost 4.1 liegt jedoch in der Art und Weise, wie GBT die Trainingsdaten behandelt. Während AdaBoost die Gewichtung aller Trainingsdaten anpasst, um die Fehler des vorherigen Lerners hervorzuheben, beschränkt sich GBT auf die Redzierung des Restfehlers.
Obwohl GBT sowohl für Klassifikations- als auch für Regressionsprobleme einsetzbar ist, konzentriert sich diese Arbeit hauptsächlich auf die Anwendung von GBT im Kontext der Regression. Der Grund dafür ist, dass GBT deutlich einfacher verständlich ist in der Art und Weise, wie er kontinuierliche Werte vorhersagt.

5.2 Die Loss-Funktion L

Ein wesentlicher Bestandteil des Gradient Boosting Trees Algorithmus ist die Loss-Funktion $L(y_i, F(x))$, welche zur Bewertung der Genauigkeit des Modells genutzt wird. Einen zentralen Teil dabei spielen die Residuen. Ein Residuum ist umgangssprachlich ein Fehler, also die Differenz zwischen einem tatsächlichen Datenpunkt y_i und dem dazu aus dem Modell vorhergesagten Wert $F(x_i)$. Im Rahmen von GBT wird häufig der Begriff "Pseudo-Residuen"verwendet. Diese werden in jeder Iteration m des Algorithmus berechnet und sind definiert als der negative Gradient der Loss-Funktion bezüglich der Modellvorhersage:

$$r_{i,m} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\bigg|F(x) = Fm - 1(x) \tag{4}$$

Im Gegensatz zu einfachen Residuen, welche den Fehler zwischen Vorhersage und tatsächlichem Wert angeben, gibt ein Pseudo-Residuum sowohl die Richtung als auch das Ausmaß der erforderlichen Anpassungen des Modells an. Bei GBT ist das besonders wichtig, da der in jeder Iteration hinzugefügte Entscheidungsbaum speziell darauf trainiert wird, diese Pseudo-Residuen zu minimieren.

5.2.1 Der Mittlere Quadratische Fehler (MSE)

Eine häufig genutze Loss-Funktion bei GBT ist der mittlere quadratische Fehler (Mean squared error, MSE). Gerade für Regressionsprobleme eignet sich die Loss-Funktion MSE besonders und ist definiert als:

$$L(y_i, F(x_i)) = \frac{1}{2}(y_i - F(x_i))^2 \tag{5}$$

Setzen wir den MSE in die Gleichung 4 für Residuen ein, ergibt sich nach Frochte [3, S. 346]

$$r_{i,m} = y_i - F_{m-1}(x_i) \tag{6}$$

Die Gleichung 6 ist zwar nicht mehr so allgemein anwendbar wie Gleichung 4, ist aber durch die Spezialisierung auf den MSE (Gleichung 5) deutlich anschaulicher und leichter zu verstehen.

5.3 Algorithmus-Struktur und Funktionsweise

Der folgende Abschnitt erklärt die einzelnen Schritte des GBT-Algorithmus (siehe Algorithmus 2):

Schritt 1: Initialisierung (Zeile 2) Zu Beginn wird ein Basismodell $F_0(x)$ erstellt. Häufig, wie auch in dem dargestellten Algorithmus 2, ist $F_0(x)$ eine Konstante, die sich aus dem Durchschnitt der Zielwerte y bildet. Der Hintergrund dafür ist, dass ein solches Basismodell einen guten Ausgangspunkt für die Optimierung bietet, da es bereits eine grundlegende Schätzung der Zielwerte liefert. Diese erste Schätzung wird dann in folgenden Iterationen verbessert.

Schritt 2: Iterationen (Zeile 3 bis 7) Der Hauptteil des Algorithmus ist die Erstellung von Entscheidungsbäumen, welchen dann dem Gesamtbaum hinzugefügt werden. Dies passiert über insgesamt M Iterationen.

Schritt 2.1: Berechnung der Residuen (Zeile 4) In jeder Iteration m werden die Residuen $r_{m,i}$ für jeden Datenpunkt i berechnet. Wie in Unterabschnitt 5.2 erklärt, zeigen letztere die Abweichung der tatsächlichen Werte y_i von den Vorhersagen des aktuellen Modells $F_{m-1}(x_i)$ auf. Die in Algorithmus 2 dafür verwendete Loss-Funktion ist der in MSE (5.)

Schritt 2.2: Training eines neuen Entscheidungsbaums (Zeile 5) In diesem Schritt wird ein neuer Entscheidungsbaum h_t trainiert, wobei der Fokus nicht auf den ursprünglichen Zielwerten der Trainingsdaten (y_i), sondern auf den Residuen $r_{m,i}$ liegt. Jeder Datenpunkt (x_i, y_i) wird also durch das Paar $(x_i, r_{m,i})$ ersetzt, wobei $r_{m,i}$ das Pseudo-Residuum ist.
Dies bedeutet konkret, dass der Entscheidungsbaum nicht darauf trainiert wird, die eigentlichen Zieldaten y direkt vorherzusagen. Stattdessen lernt der Baum, die Fehler des aktuellen Modells, also die Residuen, zu korrigieren. Dadurch das der Baum lediglich die Fehler korrigiert, ergibt es hingegen zu AdaBoost keinen Sinn ihn einzeln zu betrachten. Lediglich im Zusammenhang des Gesamtmodells ist die Hypothese relevant.

Schritt 2.4: Aktualisierung des Modells (Zeile 6) Das Modell wird in jedem Schritt verbessert, indem der neue Entscheidungsbaum $h_m(x)$ zum bisherigen Modell $F_{t-1}(x)$ mit der Gewichtung α hinzugefügt wird. Statt als Lernrate einen festen Hyperparamter α zu benutzen, gibt es auch wie in Frochte [3, S. 345] die Möglichkeit diesen dynamisch zu berechnen. Meist dann mit γ bezeichnet ist der Wert meist für jeden Baum unterschiedlich. Für die Suche nach γ kann beispielsweise Linearsuche verwendet werden.

Während die Verwendung von α bedeutet, dass jeder Baum gleichgewichtet ist, lässt sich mit γ feinere und damit auch effizientere Anpassungen am Gesamtmodell machen. Dies ist natürlich aber auch mit einem höheren Rechenaufwand verbunden.

Schritt 3: Ausgabe des finalen Modells (Zeile 8) Nach M Iterationen steht das finale Modell $F_M(x)$ fest. Es fasst die kumulativen Verbesserungen aller Entscheidungsbäume zusammen und vereint damit alle Verbesserungen zur Anpassung auf die Zieldaten y.

Algorithm 2 MSE Gradient Tree Boosting (nach Frochte [3, S. 346])

1: **Gegeben:** Trainingsdatensatz $(x_1, y_1), \ldots, (x_n, y_n)$
2: **Initialisierung:** Starte mit einem Basismodell $F_0(x)$, z.B. dem Mittelwert der Labels y
3: **for** $m = 1, \ldots, M$ **do**
4: Berechne die Pseudo-Residuen $r_{i,m} = y_i - F_{m-1}(x_i)$ für alle i
5: Trainiere einen Entscheidungsbaum h_m auf $(x_i, r_{i,m})_{i=1}^n$
6: Führe das Update durch: $F_m(x) = F_{m-1}(x) + \alpha \cdot h_m(x)$
7: **end for**
8: **Ausgabe:** Das endgültige Modell $F_M(x)$

5.4 Beispielanwendung mit Erläuterung

Um den doch eher abstrakten GBT-Algorithmus 2 zu veranschaulichen, wird im folgenden die Funktionsweise anhand des Beispiels in Abbildung 5 beschrieben.
Die Visualisierung, sowie die verwendeten Daten wurden mithilfe von SKLearn in python erstellt und sind auf meinem GitHub zu finden. Wir betrachten den folgenden synthetischen Datensatz als gegeben:

```
X, y = make_regression(n_samples=100, n_features=1, noise=10, random_state=42)
X_sorted, X_indices = np.sort(X, axis=0), np.argsort(X, axis=0)
```

Wie im oberen Bild der Abbildung 5 zu erkennen ist, wird im Schritt der Initialisierung zunächst das Basismodell aus dem Durchschnitt der Zielwerte gebildet. Wie zu erwarten, hat das Basismodell keine wirkliche Aussage über die Zieldaten mit einem MSE von 1690.

In der ersten Iteration wird dann eine erste Vorhersage auf die Daten getroffen. Für diese Iteration stehen natürlich noch keine Residuen des Vorgängermodells zur Verfügung, weshalb sich die Punkte der Daten und Residuen decken. Schon nach einer Iteration nimmt die Vorhersage die ungefähre Form der Daten an. Alleine in diesem Schritt kann der MSE um das 5-fache reduziert werden.
Hier lässt sich zum ersten mal das Prinzip der Residuen veranschaulichen. Nehmen wir als Beispiel den einzelnen Datenpunkt in der unteren linken Ecke. Die Gesamtvorhersage der ersten Iteration hat eine Abweichung von ca. 80. Genau diese Abweichung von ca. 80 spiegelt sich dann für die zweite Iteration bei dem Residuum in der unteren linken Ecke wieder. In der zugehörigen Baumvorhersage lässt sich auch erkennen, dass sie deutlich besser auf die Residuen

passt als die Ursprungsdaten. Trotzdem lässt sich in diesem Schritt der MSE fast halbieren. Dies veranschaulicht die Argumentation aus Schritt 2.2.

Nach der dritten Iteration entspricht die Gesamtvorhersage ziemlich genau den Daten mit einem verbleibenden MSE von 89. Die Schrittweise Verbesserung des Modells lässt sich neben dem Offensichlichen auch durch eine weitere Beobachtung erklären. In jeder Iteration nähren sich die Residuen der X-Achse an. Nach einer unbekannten Iteration von Schritten ist davon auszugehen, dass wenn die Gesamthypothese die Daten perfekt abbildet, sich alle Residuen auf der X-Achse befinden.

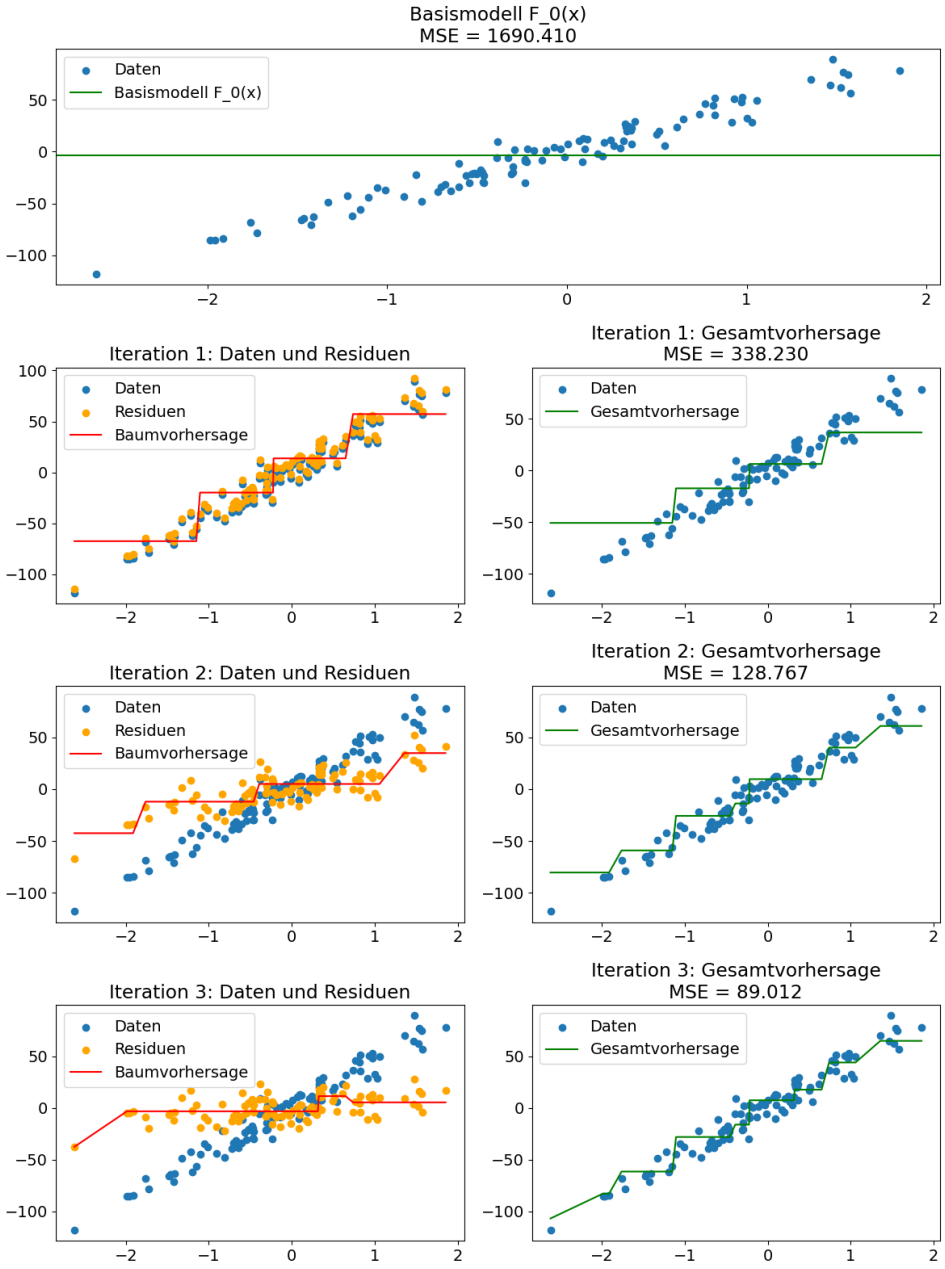

Abbildung 5: Visualisierung des Gradient Boosting Trees Algorithmus.

6 Vergleich von AdaBoost und Gradient Boosting (4-5 Seiten)

6.1 Gemeinsamkeiten und Unterschiede

6.2 Performance-Analyse in Benchmarks

6.3 Anwendungsbeispiele und Fallstudien

!!! REFERENZ ZU 1. EINLEITUNG

7 Aktuelle Trends und Entwicklungen (2-3 Seiten)

7.1 Neueste Forschungsergebnisse

7.2 Zukünftige Potenziale von Boosting-Algorithmen

7.3 Toleranz gegen Overfitting

!!! VERWEISEN

7.4 Exponentieller Abstieg des Trainingfehlers

!!! VERWEISEN

8 Fazit und Ausblick (2-3 Seiten)

8.1 Zusammenfassung der Erkenntnisse

8.2 Reflexion über die Bedeutung für die Praxis

8.3 Ausblick auf zukünftige Forschungsthemen

Literaturverzeichnis

[1] R. E. Schapire und Y. Freund, *Boosting : foundations and algorithms* (Adaptive Computation and Machine Learning Series). Cambridge, Mass. [u.a.]: MIT Press, 2012, Includes bibliographical references and index, ISBN: 9780262017183. Adresse: http://www.gbv.de/dms/ilmenau/toc/672276232.PDF%20;%20https://zbmath.org/?q=an:1278.68021.

[2] A. [Géron, *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow : Konzepte, Tools und Techniken für intelligente Systeme*, 1. Auflage, K. [Rother, Hrsg. Heidelberg: OReilly, 2018, Äuthorized German translation of the English edition of 'Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems', ISBN 978-1-491-96228-9, (c) 2017 Impressum; ÜS-Bestseller zu Deep Learning Umschlag"Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint Ö'Reilly". Rückseite der Titelseite. - "Deutschsprachige O'Reilly-Bücher werden vom dpunkt.verlag in Heidelberg publiziert, vermarktet und vertrieben.- Webseite www.oreilly.de; 201712; aa, ISBN: 9783960090618. Adresse: http://www.gbv.de/dms/ilmenau/toc/898831717.PDF%20;%20http://vub.de/cover/data/isbn%3A9783960090618/medium/true/de/vub/cover.jpg%20;%20http://deposit.d-nb.de/cgi-bin/dokserv?id=6ba05f7dfc5242d6bbc6f653e199e877&prov=M&dok_var=1&dok_ext=htm%20;%20https://www.oreilly.de/buecher/13111/9783960090618-praxiseinstieg-machine-learning-mit-scikit-learn-und-tensorflow.html.

[3] J. [Frochte, *Maschinelles Lernen : Grundlagen und Algorithmen in Python* (Hanser eLibrary), 3., überarbeitete und erweiterte Auflage. München: Hanser, 2020, ISBN: 9783446463554. Adresse: https://dx.doi.org/10.3139/9783446463554%20;%20https://doi.org/10.3139/9783446463554%20;%20https://www.hanser-elibrary.com/doi/book/10.3139/9783446463554%20;%20http://dx.doi.org/10.3139/9783446463554.

[4] G. [James, *An introduction to statistical learning : with applications in Python* (Springer texts in statistics), D. [Witten, T. [Hastie, R. [Tibshirani und J. E. [Taylor, Hrsg. Cham, Switzerland: Springer, 2023, ISBN: 9783031387463.

[5] T. [Hastie, *The elements of statistical learning : data mining, inference, and prediction* (Springer series in statistics), Second edition, R. [Tibshirani und J. H. [Friedman, Hrsg. New York, NY: Springer, 2009, Hier auch später erschienene, unveränderte Nachdrucke; Literaturverzeichnis: Seite 699-727, ISBN: 9780387848570. Adresse: http://www.gbv.de/dms/ilmenau/toc/572093853.PDF%20;%20https://zbmath.org/?q=an:1273.62005%20;%20https://swbplus.bsz-bw.de/bsz287727726kap.htm%20;%20https://swbplus.bsz-bw.de/bsz287727726vor.htm%20;%20https://swbplus.bsz-bw.de/bsz287727726inh.htm%20;%20https://swbplus.bsz-bw.de/bsz287727726cov.jpg.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis