Technische Hochschule Deggendorf

Fakultät Angewandte Informatik

(Bachelor-Studiengang Angewandte Informatik)

Ensemble Learning

Seminararbeit für den Kurs

Grundlagen des wissenschaftlichen Arbeitens 2

an der Technischen Hochschule Deggendorf

vorgelegt von:

Sage, Sandro

00805804

am: 31. Dezember 2022

Prüfer∗in:

Prof. Dr. Kristina Wanieck

Abstrakt

Hier beginnt das Abstract.

Inhaltsverzeichnis

Gleichungsverzeichnis

Abbildungsverzeichnis

Akronyme

CART	Classification and regression trees. 2
CV	Cross-Validation. 4
KI	Künstliche Intelligenz. 1
KNN	K-nearest neighbors. 3
ML	Machine Learning. 1, 7

Glossar

Base Learner Basislerner - individuelle Lernalgorithmen in homogenen Ensembles. 1, 2

Label (Klassen)-Bezeichnung - gebräuchlicher englischer Ausdruck. 2

Output Ausgabe - herkömmlicher englischer Begriff. 3, 4

Sample Datenpunkt - gebräuchlicher englischer Ausdruck. 1

1. Einleitung

In der heutigen Zeit nimmt die Popularität und Anwendungsmöglichkeit künstlicher Intelligenz immer stärker zu [2, S.1] Dies liegt einerseit an der Explosion erstellter und öffentlich zugänglicher Daten, aber auch anderseits an der vorgeschrittenen Forschung im Bereich *Machine Learning (ML)*, welcher als Teilbereich der *Künstliche Intelligenz (KI)* fungiert. Zudem tragen die erweiterten „computing capabilities“ [2, S.1] zu neuen Möglichkeiten der Applikation von *ML* bei. Ein komplexeres und fortgeschrittenes *ML*-Thema ist das *Ensemble Learning*. Im folgenden soll dieses komplexe Thema hinsichtlich der allgemeinen Definition, der verschiedene Methodenansätze, der Aggregationsmöglichkeiten und der Anwendungsbereiche dargestellt und erläutert werden. Unter *Ensemble Learning* versteht man ein „multiple classifier system“[7, S.182], welches auch „committee-based learning“[6, S.15] genannt wird. Hierzu werden individuelle Lernalgorithmen zunächst auf bestimmte Art und Weise trainiert und folgend mit Hilfe einer bestimmten Kombinationstechnik aggregiert. Es gibt zwei verschiedene Arten, den homogenen und heterogenen Ensembles, zwischen denenen unterschieden werden kann. Homogen bedeutet, dass alle individuellen *Base Learners*[7, S.182] auf den gleichen Lernalgorithmen basieren, wodurch hingegen heterogene Ensembles auf die Verwendung unterschiedlicher *base learners* und somit Lernalgorithmen aufbauen. Werd ein heterogenes Ensemble betrachtet, so spricht man jedoch nicht mehr von *base learners*, sondern von „component learners“[6, S.15]. Ein Vorteil bei der Anwendung von *Ensemble Learning*-Methoden ist die Verbesserung des Verallgemeinerungsfähigkeit, besser bekannt unter der „generalization ability“[7, S.15], wobei damit die Fähigkeit gemeint ist, wie gut der Lernalgorithmus nicht nur die Trainigsdaten, sondern auch noch nie zuvor gesehene Testdaten generalisieren kann.

2. Bagging

Da die Verallgemeinerungsfähigkeit der Ensembles von der Unabhängigkeit der individuellen *base learners* abhängt und eine strikte Unabhängigkeit leider nicht gewährleistet werden kann, wird versucht, die Lernalgorithmen so unterschiedlich wie möglich voneinander zu erstellen. Dies kann bereitgestellt werden, indem jeder *base learner* auf einer bestimmten Partition des ganzen, originalen Datensatzes trainiert wird. Dabei muss beachtet werden, dass wenn nicht ausreichend Daten für das Training zur Verfügung stehen, die Partitionen so erstellt werden müssen, dass sie sich in gewissen *Samples* überlappen [7, vgl. S.189]. Eine Ensemble Methode die diesen Ansatz verfolgt, ist *Bagging*[3]. *Bagging* ist eine repräsentative Methode des parallelem

Ensemble Learning's, die auf dem sogenannten *Bootstrap sampling* basiert. Der Prozess funktioniert folgendermaßen: Gibt es einen Datensatz $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, wobei y_i dem *Label* des Datensamples entsptricht, mit m-Samples, dann wird „randomly"[7, S.190] ein Sample aus dem Datensatz kopiert. Dieser Vorgang wird mit „replacement"[3, S.1] ausgeführt, dass heißt, dass eben nur das Sample kopiert wird und sich danach noch immer im originalen Datensatz befindet. Führt man diesen Prozess T-mal durch so entstehen dementsprechend auch T Datensatzpartitionen. Die *Base Learner* können nun auf den Partitionen trainiert und daraufhin kombiniert werden [7, vgl. S.190]. Eine Übersicht des Verfahren bietet Abbildung 1. Der Aspekt des *replacements* wird durch die orangen Samples veranschaulicht, da eigentlich nur ein oranges Samples in D enthalten ist, jedoch durch die *bootstrap sampling*-Methode das orange Sample mehr als einmal in den Bootstrapdatensätzen $\{D_{B1}, D_{B2}, \ldots, D_{BT}\}$ existieren kann (siehe D_{B2}, D_{BT}).

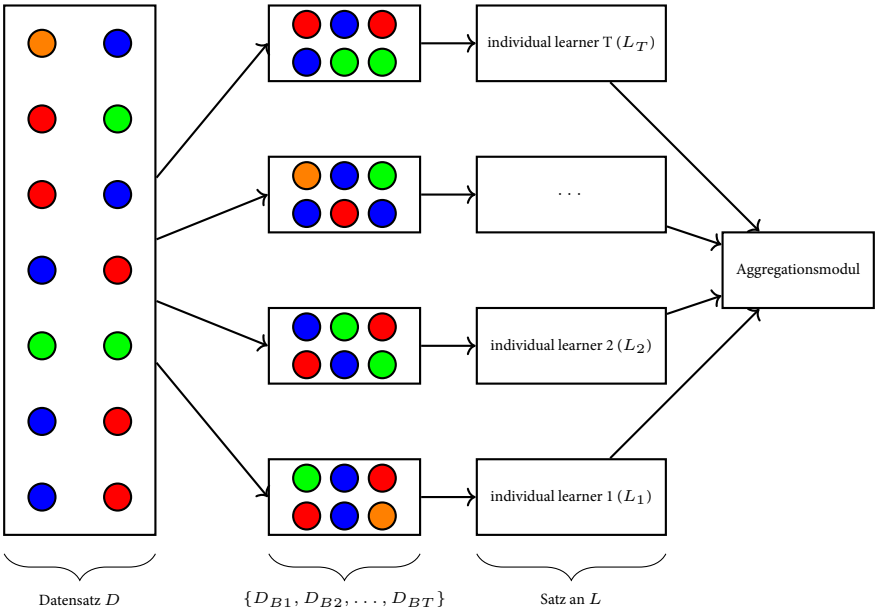

Abbildung 1: Vorgehensweise der *Bagging*-Methode mittels *Bootstrap sampling*. Das Traning der *individual learners* wird durch die parallele Architektur des System dargestellt.

Eine Bedingung für die Funktionsweise der *Bagging*-Methode ist die Instabilität der Lernalgorithmen, wobei Breiman[3] diese Vorraussetzung studiert hat und es sich herausstellte, dass z.B. Neuronale Netze, *Classification and regression trees (CART)* eine gewisse Instabilität

aufweisen währenddessen der *K-nearest neighbors (KNN)*-Algorithmus stabil bleibt. Gibt es *base learners*, die unempfinlich gegenüber „perturbation"[6, S.51] in den Trainingssamples sind und somit gewissermaßen gleich sind, dann wird eine Kombination dieser *base learners* keine Verbesserung in der Verallgemeinerung gewährleisten. Diese *base learners* nennt man dann auch „stable learners"[6, S.51]. Breiman[3] testete dies anhand sechs verschiedener Datensätze und stellt den Beweis hierfür in Tabelle 1 dar. Ein Vorteil des *Bagging* Ensemble Learning's ist die Parallelität, die es ermöglicht,durch Verwendung von „multi-core computing processors"[7, S.48] oder parallelen Computerarchitekturen, die Trainingszeit zu minimieren.

Tabelle 1: Vergleich der K-nearest neighbors-Klassifizierungsfehler mit $(\bar{e}_B)$ und ohne $(\bar{e}_S)$ Bagging. Es fällt sofort auf, dass sich der Klassifierungsfehler kaum bzw. gar nicht verringert. Dieser Vergleich wurde anhand sechs verschiedener Datensätzen erstellt [3, vgl.S.14].

Datensatz	$\bar{e}_S$	$\bar{e}_B$
waveform	26.1	26.1
heart	5.1	5.1
breast cancer	4.4	4.4
ionosphere	36.5	36.5
diabetes	29.3	29.3
glass	30.1	30.1

3. Stacking

Die zweite Methode des *Ensemble Learning*'s ist das sogenannte *Stacking*. Wenn hier ausreichend viel Daten vorhanden sind, können unter der Verwendung eines *meta-learners* mehrere individuelle Lerner kombiniert werden. Es wird auch von „[C]ombining by learning"[7, S.196] gesprochen, da das Aggregationsmodul selbst durch einen individuellen Lerner aufgebaut wird. Die individuellen Lerner werden *first-level learners* und die Lerner, die die Kombination ausüben, *second-level learners* oder *meta learners* genannt. Im ersten Schritt des *Stacking*'s werden die *first-level learners* auf Basis des originalen, gesamten Datensatzes trainiert und im nächsten Schritt wird ein neuer Datensatz, der dann als Trainingssatz für die *meta-learners* fungiert, generiert. Der neu-generierte Datensatz besteht aus den Vorhersagen oder *Outputs* der *first-level learners*. Der generierte Datensatz darf nicht exakt auf Basis der Trainingssamples im vorherigen Schritt erstellt werden, da sonst ein hohes Risiko an *Overfitting* besteht. Deshalb wird in diesem Ansatz das *k-fold Cross-Validation* angewendet. Beim *k-fold CV* wird der

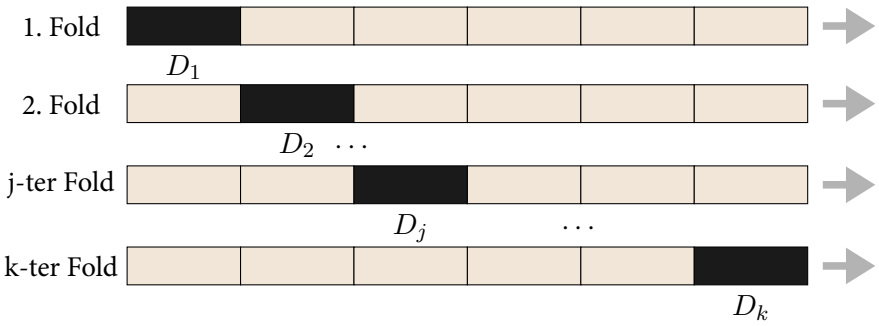

Abbildung 2: Veranschaulichung der *K-fold Cross Validation* Partitionsaufteilung. Hier entspricht D_j dem Testdatensatz und $\bar{D}_j$ dem Trainingsdatensatz.

originale Datensatz in k annähernd gleich große Partitionen $D_1, D_2, \ldots, D_k$ unterteilt. D_j wird als Testsatz und $\bar{D}_j = D/D_j$ als Trainingssatz des j-ten Folds deklariert. Wenn es T *first-level learners* gibt und $h_t^{(j)}$ den t-en Lernalgorithmus, der auf Basis des Trainingssatzes $\bar{D}_j$ trainiert wurde, entspricht, dann ist $z_{it} = h_t^{(j)}(x_i)$ für jedes Sample x_i in D_j der zugehörige *Output*. Diese *Outputs* der *first-level learners* kann durch $z_i = (z_{i1}; z_{i2}; \ldots; z_{iT})$ ausgedrückt werden [7, vgl. S.197, 29] [6, vgl. S84f.]. Daraufhin werden die *Outputs* in Verbindung mit den originalen Annotationen y_i als Eingabe für den *meta learner* verwendet. Schlussendlich entsteht durch den vollständigen Prozess des *Cross-Validation (CV)* der neu generierte Datensatz $D' = \{(z_i, y_i)\}_{i=1}^{m}$, der für das Training des *second-level learners* angewendet werden kann. Eine vereinfachte Darstellung der eben geschilderten Datenaufteilung wird in Abbildung 2 gezeigt.

4. Boosting

Die letzte Methoden ist das sognannte *Boosting*. Bei diesem Verfahren wird versucht „weak learners“[7, S.184] in „strong learners“[7, S.184] umzuwandeln. Zunächst wird mit dem Training des ersten *base learners* auf Basis einer bestimmten Datenverteilung begonnen. Danach werden die Trainingssamples hinsichtlich des Ergebnisses der ersten Trainingsphase angepasst. Dies passiert, indem Samples, die als falsch klassifiziert wurden, nun mehr Aufmerksamkeit in den noch folgenden Trainingsphasen der weiteren *base learners* bekommen. Vereinfacht bedeutet das, dass nach dem Training des ersten *base learner*'s, die Datenverteilung angepasst wird und daraufhin der zweite *base learner*, dessen Ergebnis für die Anpassung der Datenverteilung für den nächsten Iterationsschritt verwendet wird, traniert wird. Der Prozess wiederholt sich

so lange bis eine vordefinierte Anzahl T an *base learners* erreicht wird. Schlussendlich erhält dann jeder dieser Lernalgorithmen eine Gewichtung und wird zusammen mit den anderen kombiniert. Eine der bekanntesten und einflussreichsten Boosting-Algorithmen ist der *AdaBoost*-Algorithmus [5]. Der Algorithmus setzt voraus, dass der gewählte *base learner* in der Lage ist, mit bestimmten Datenverteilungen trainiert werden zu können. Eine Möglichkeit für das Erreichen einer solchen Datenverteilung bietet das *re-weighting*, bei dem nach jeder Trainingsrunde ein neues Gewicht den Samples in der Datenverteilung zugeordnet wird. Eine weitere Option bietet das *re-sampling*, welches verwendet werden kann, falls der Lernalgorithmus *re-weighting* nicht unterstützt. Hier wird in jeder Trainingsinteration eine neue Teilmenge aus der gesamten Datenverteilung entnommen [6, vgl. S.23ff.] [7, vgl. S.184/S.188]. Eine Erweiterung des *re-sampling*'s bietet den Neustart des *Boosting*'s an, wobei dieser Ansatz in *Bias plus variance decomposition for zero-one loss functions* [1] genauer erläutert wird.

5. Kombinierungsansätze

Nachdem nun verschiedene Methoden des *Ensemble Learning*'s vorgestellt worden sind, behandelt dieses Kapitel die Kombinierungsoptionen nach denen die *base learners* zusammengefügt werden können. Dietterich[4, vgl. S.3f.] liefert drei Perspektiven, die erkläutern warum das Kombinieren vorteilhaft ist. Aus statistischer Sicht ist der Hypothesenraum meistens sehr groß, wodurch es mehrere Hypothesen, die die gleichen Ansätze anstreben, gibt. Wenn nur ein einzelner Lernalgorithmus oder *base learner* betrachtet wird, dann hängt die Verallgemeinerungsgenauigkeit (*generalization performance*) ausschließlich von der Genauigkeit dieses *base learners* ab. Werden nun mehrere Hypothesen kombiniert, so verringert sich auch das Risiko einer Falschaussage. Werden die Lernalgorithmen aus rechnerischer Perspektive betrachtet, so können diese oftmal in sogenannten „local optimum[s]"[7, S.193] feststecken. Werden jedoch mehrere Lernalgorithmen verwendet, so senkt sich auch hier wieder das Risiko in einem *local optimum* festzustecken. Zuletzt gibt es noch die repräsentative Perspektive, bei der angenommen wird, dass die richtige Hypothese nicht in Raum der existierenden Hypothesen befindet. Somit kann durch Verwendung mehrer Hypothesen eine genauere Annäherung an die richtige Hypothese gewährleistet werden [7, vgl. S.194] [4, vgl. S.4]. In den weiteren Unterkapiteln werden die verschiedenen Strategien der Kombination erläutert.

5.1. Averaging

Die erste Art der Kombination ist das *Averaging*, die in der Regression ihre Anwendung findet, da hier ein nummerischer Ouput erwartet wird. Typische *Averaging*-Methoden sind das *simple Averaging*

$$H(x) = \frac{1}{T} \sum_{i=1}^{T} h_i(x) \tag{1}$$

und das *weighted Averaging*

$$H(x) = \frac{1}{T} \sum_{i=1}^{T} w_i h_i(x) \tag{2}$$

Es wird angenommen, dass das Ensemble T individuelle Lerner bzw. *base learners* $\{h_1, h_2, \ldots, h_T\}$ beinhaltet, wobei $h_i(x)$ dem Output des i-ten *base learners* auf das Sample x entspricht. Bei dem *weighted Averaging* steht w_i für das Gewicht des i-ten individuellen Lerners. Typischerweise gilt $w_i \geqslant 0$ und $\sum_{i=1}^{T} w_i = 1$. Viele empirische Studien haben bewiesen, dass *weighted Averaging* nicht notwendigerweise besser abschneidet als *simple Averaging* FEHLENDE QUELLEN IN ML. Wenn die einzelnen *base learners* ähnliche Genauigkeit aufweisen, so gilt im Allgemeinen, dass *simple Averaging* zu verwenden. Weisen die individuellen Lerner jedoch erhebliche Unterschiede in der Genauigkeit auf, so wird das *weighted Averaging* vorgezogen [7, vgl. S.195].

5.2. Voting

Da nun auf die Anwendung für die Regression beschrieben wurde, befasst sich dieser Abschnitt mit den Methoden für die Klassifizierung. Hierfür wird das *Voting*, dass in drei verschiedenen Arten vorgestellt wird. Bei dem *Voting* gibt es individuelle Lerner $\{h_1, h_2, \ldots, h_T\}$ und h_i kann N verschiedene Klassenlabels $\{c_1, c_2, \ldots, c_N\}$ vorhersagen. Entspricht der N-dimensionale Vektor $(h_i^1(x), h_i^2(x), \ldots, h_i^N(x))$ und h_i dem Output des Samples x, so ist $h_i^j(x)$ gleich der Output von h_i für das Klassenlabel c_j. Die erste *Voting*-Methode ist das *Majority Voting*

$$H(x) = \begin{cases} c_j, & \text{if} \sum_{i=1}^{T} h_i^j(x) > 0.5 \sum_{k=1}^{N} \sum_{i=1}^{T} h_i^k(x); \\ \text{reject}, & \text{otherwise.} \end{cases} \tag{3}$$

,die zweite Methode ist das *Plurality Voting*

$$H(x) = c_{\arg\max_j \sum_{i=1}^{T} h_i^j(x)} \tag{4}$$

und zuletzt das *Weighted Voting*

$$H(x) = c_{\arg\max_j \sum_{i=1}^{T} w_i h_i^j(x)} \tag{5}$$

. Wird das *Weighted Voting* geanuer betrachtet so lässt sich erkennen, dass es vom *Plurality Voting* abgeleitet ist und als Erweiterung noch Gewichtungen berücksichtigt werden. w_i entspricht somit dem Gewicht von h_i. Ein Nachteil des *Majority Voting*'s ist die *reject*-Option, da in vertrauenswürdigen Anwendungen, wie z.B. in der Medizin, immer ein Resultat getroffen werden muss [7, vgl. S.195]

6. Anwendungsbereiche

Es gibt mehrere Anwendungsbereiche in denen *Machine Learning* durch *Ensemble Learning* erweitert werden kann für eine deutliche Verbesserung der Genauigkeit und Performanz sorgt. Ein Beispiel ist der KDD-Cup[1], der einer der bekanntesten Data-Mining Wettbewerb entspricht. Die verschiedenen Wettbewerbsaufgaben über die Jahre waren z.B. Erkennung von Netzwerkeinbrüchen (1999), molekulare Bioaktivität und Vorhersage von Proteinstandorten (2001) und Musikempfehlung (2011). Für das Lösen der Aufgaben wurden verschiedene Techniken verwendet, wobei *Ensemble Learning* am meisten an Aufmerksamkeit gewann. In den Jahren 2009 bis 2011 benutzten die erst- und zweitplazierten Gewinner verschiedene *Ensemble Learning*-Methoden. Ein weiteres Beispiel bietet der *Netflix Prize*, der von dem Streamingdienstanbieter Netflix im Jahr 2009 verliehen wurde. Ziel war es Voraussagen treffen zu können, wie sehr jemand einen Film aufgrund seiner Vorlieben genießen wird. Den 1M Dollar Preis gewann dann das Team *BellKor's Pragmatic Chaos*, da ihr Lösungsansatz auf der Kombination verschiedenen Lernalgorithmen basierte und ihnen somit zum Sieg verhalf.

[1] `https://kdd.org/kdd-cup`

Im Allgemeinen kann *Ensemble Learning* erfolgreich in den verschiedensten Bereichen eingesetzt werden. Weitere Beispiele hierfür sind *Computer Vision*, *Object Detection*, Erkennung von Kreditkartenbetrug und Wettervorhersagen [6, vgl. S.17f.].

Literaturverzeichnis

[1] *Bias plus variance decomposition for zero-one loss functions*. 1996. URL: http://robotics.stanford.edu/users/ronnyk/biasvar.pdf (siehe Seite 5).

[2] R. Boutaba et al. „A comprehensive survey on machine learning for networking: evolution, applications and research opportunities“. In: *Journal of Internet Services and Applications* 9.1 (2018), S. 1–99. ISSN: 1867-4828. DOI: 10.1186/s13174-018-0087-2. URL: https://link.springer.com/article/10.1186/s13174-018-0087-2 (siehe Seite 1).

[3] L. Breiman. „Bagging predictors“. In: *Machine Learning* 24.2 (1996), S. 123–140. ISSN: 0885-6125. DOI: 10.1007/BF00058655. URL: https://link.springer.com/article/10.1007/BF00058655 (siehe Seiten 1–3).

[4] T. G. Dietterich. „Ensemble Methods in Machine Learning“. In: Bd. 1857, S. 1–15. DOI: 10.1007/3-540-45014-9_1 (siehe Seite 5).

[5] Y. Freund und R. E. Schapire. „A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting“. In: *Journal of Computer and System Sciences* 55.1 (1997), S. 119–139. ISSN: 00220000. DOI: 10.1006/jcss.1997.1504 (siehe Seite 5).

[6] Z.-H. Zhou. *Ensemble Methods*. Chapman and Hall/CRC, 2012. ISBN: 9780429151095. DOI: 10.1201/b12207. URL: https://www.taylorfrancis.com/books/mono/10.1201/b12207/ensemble-methods-zhi-hua-zhou (siehe Seiten 1, 3–5, 8).

[7] Z.-H. Zhou. *Machine Learning*. Singapore: Springer Singapore, 2021. ISBN: 978-981-15-1966-6. DOI: 10.1007/978-981-15-1967-3 (siehe Seiten 1–7).

A. AdaBoost

Algorithmus 1: AdaBoost

Eingabe: Trainingsdatensatz $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$;
Base Learner $\mathcal{L}$;
Anzahl der Trainingsrunden T.

Beginn
  $\mathcal{D}_1(\boldsymbol{x}) = 1/m$;
  für $t = 1, 2, \ldots, T$ **tue**
    $h_t = \mathcal{L}(D, \mathcal{D}_t)$;
    $\epsilon_t = P_{\boldsymbol{x}\ D_t}(h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}))$;
    wenn $\epsilon_t > 0.5$ **dann**
      abbrechen
    Ende
    $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$;
    $$\mathcal{D}_{t+1}(\boldsymbol{x}) = \frac{\mathcal{D}_t(\boldsymbol{x})}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{wenn } h_t(\boldsymbol{x}) = f(\boldsymbol{x}); \\ \exp(\alpha_t), & \text{wenn } h_t(\boldsymbol{x}) \neq f(\boldsymbol{x}); \end{cases}$$
    $$= \frac{\mathcal{D}_t(\boldsymbol{x}) \exp(-\alpha_t f(\boldsymbol{x}) h_t(\boldsymbol{x}))}{Z_t};$$
  Ende
Ende

Ausgabe: $F(\boldsymbol{x}) = sign(\sum_{t=1}^{T} a_t h_t(\boldsymbol{x}))$

Kolophon

Dieses Dokument ist ein LaTeX3 Dokument der KOMA-Script Klasse und wurde mit *Minion Pro* gesetzt. Alle eigenen Zeichnungen sind mit Ti*k*Z und PGFPLOTS erstellt. Kompiliert wurde es mit LuaLaTeX und biber auf Windows am 31. Dezember 2022.