Ausarbeitung zum Thema AdaBoost

Bachelor-Seminar „Top 10 Algorithms in Data Mining“

Marius Graf

12. Dezember 2023

Inhaltsverzeichnis

Abstract: Diese Arbeit beschäftigt sich mit dem AdaBoost-Algorithmus, einer Ensemble-Methode im maschinellen Lernen, die mehrere schwache Lerner kombiniert, um ein präzises Gesamtmodell zu erstellen. Die Arbeit erklärt die Grundlagen des Boosting-Konzepts, den Ablauf des AdaBoost-Algorithmus und umreißt seine Anwendung in der Praxis. Zuletzt werden auch die Vor- und Nachteile von AdaBoost diskutiert sowie einige Erweiterungen und Variationen des Algorithmus für unterschiedliche Problemstellungen dargestellt.

1 Einleitung

Data Mining analysiert große Datenmengen, um Muster und Zusammenhänge zu erkennen, wobei Methoden aus Statistik, Machine Learning und Datenbanktechnologie eingesetzt werden. Es spielt eine zentrale Rolle in Forschung und Industrie, um Erkenntnisse zu gewinnen und Entscheidungen zu unterstützen. AdaBoost gehört zu den *Ensemble-Methoden*, die mehrere Modelle kombinieren, um präzisere Vorhersagen zu treffen. Diese Arbeit konzentriert sich auf AdaBoost, der Beobachtungen gewichtet, um zuvor falsch bzw. schlecht vorhergesagte Datenpunkte besser vorherzusagen.

2 Grundlagen des Boosting

Boosting ist eine Ensemble-Technik im Machine Learning, die mehrere sog. *„schwache Lerner“* kombiniert, um ein präziseres Gesamtmodell zu erstellen. Als *schwacher Lerner* wird ein Modell bezeichnet, das nur einen geringen Zusammenhang in den Daten lernen kann und dessen Vorhersagen daher nur etwas besser als zufälliges Raten sind. In jeder Iteration wird ein solches Modell trainiert, das versucht, falsche Vorhersagen der vorherigen Modelle zu korrigieren, indem es die Gewichtung der Trainingsdaten anpasst. Das Ziel ist es, den systematischen Fehler (*Bias*) des Modells zu reduzieren, der durch die einseitige Betrachtung der Daten entsteht. Dadurch wird in de Regel eine bessere Generalisierbarkeit, also Genauigkeit auf ungesehenen Daten erhöht.

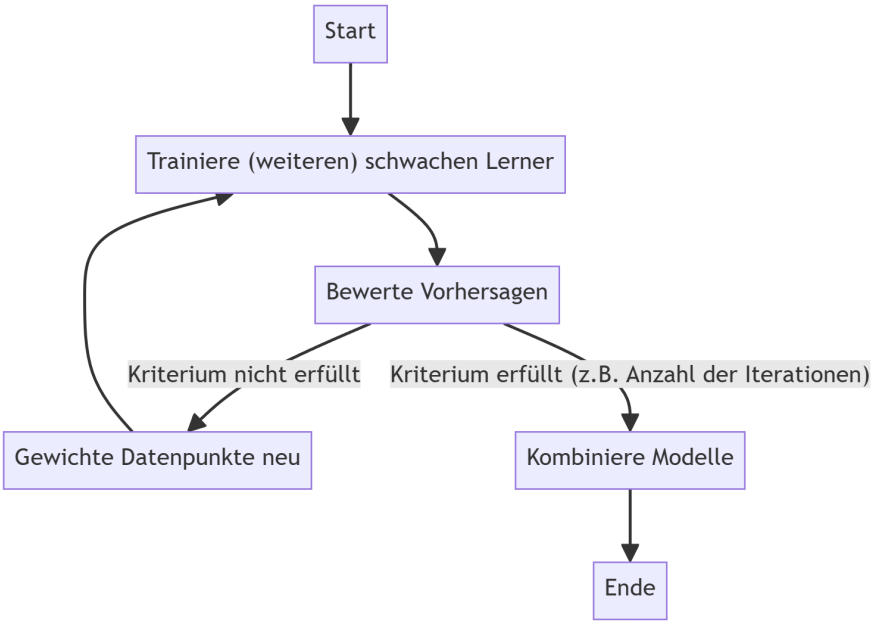


Abbildung 1: Flussdiagramm zur Veranschaulichung des Boosting-Konzepts

Beispiel:

(P(Iteration) = Vorhersage, W(Iteration) = Gewichtung)

Haus	**Größe** $[m^2]$	**Lage**	**Preis**
Haus 1	100	Zentrum	440.000€
Haus 2	150	Vorort	500.000€
Haus 3	80	Zentrum	400.000€
Haus 4	120	Ländlich	200.000€

Haus	**P(1)**	**W(1)**	**P(2)**	**W(2)**	**P(3)**
Haus 1	450.000€	0.1307	445.000€	0.1192	443.000€
Haus 2	350.000€	0.5299	495.000€	0.4833	501.000€
Haus 3	380.000€	0.1444	430.000€	0.1691	410.000€
Haus 4	250.000€	0.1950	230.000€	0.2283	205.000€

Beispielhafte Anpassung der Gewichte über die Iterationen

3 Der AdaBoost Algorithmus

Einführung

AdaBoost, kurz für „Adaptive Boosting“, wurde in den 1990ern von Yoav Freund und Robert Schapire für die binäre Klassifikation konzipiert und hat das Feld des maschinellen Lernens beeinflusst. Während Boosting-Methoden allgemein iterativ arbeiten und Fehler der vorherigen Modelle korrigieren, zeichnet sich AdaBoost durch seine spezielle Methode zur Gewichtungsanpassung der Datenpunkte aus. In jeder Trainingsiteration erhöht AdaBoost gezielt die Gewichtung der falsch klassifizierten Datenpunkte exponentiell abhängig davon, wie gut der Lerner die Daten insgesamt vorhersagt. Dieser adaptive Ansatz zur Fehlerkorrektur unterscheidet AdaBoost von anderen Boosting-Methoden und hat die Effizienz von Boosting-Methoden für binäre Klassifikationsprobleme gezeigt und viele Weiterentwicklungen in diesem Bereich angeregt. [WK09]

Notation

- Sei $\mathcal{X}$ die Menge der **Features**. $\mathcal{Y}$ die Menge der **Labels**, die gelernt werden sollen. Dabei ist $\mathcal{Y} = \{-1, +1\}$ bei binärer Klassifikation. Ein **Trainingdatensatz** D besteht aus m Einträgen, welche Features mit Labels verbinden:

$$D = \{(\boldsymbol{x}_i, y_i)\},\ i = 1, \ldots, m$$

- Nach dem Training auf D gibt ein **Lernalgorithmus** (meistens *Decision Stump*) $\mathcal{L}$ eine **Hypothese** bzw. einen Klassifizierer h zurück, der von X nach $\mathcal{Y}$ abbildet. In der Regel wrid sogar eine Folge von Hypothesen $(h_j)_{j \in \mathcal{I}}$ mit $\mathcal{I} = \{1, .., n\}$ zurückgegeben.

$$h_j : X \to \mathcal{Y},\ h_j(\boldsymbol{x}) = y$$

- T ist die Anzahl der gewünschten **Trainingsiterationen**.
- Bei jeder Iteration $t = 1, \ldots, T$ wird der Datensatz um **Gewichte**

$$w_i^{(t)},\ i = 1, .., m$$

erweitert.

Initialisierung der Gewichte

Zu Beginn sind die Gewichte aller m Datenpunkte gleich verteilt:

$$w_i^{(1)} = \frac{1}{m},\ i = 1, \ldots, m$$

Zudem ist die Summe aller Gewichte bei jeder Iteration stets 1.

$$\sum_{i=1}^{m} w_i^{(t)} = 1 \quad \forall t = 1, \ldots, T$$

Training der schwachen Lerner

Trainiere in jeder Iteration t eine Folge von schwachen Lernern unter Berücksichtigung der aktuellen Gewichtung.

$$(h_j)_{j \in \mathcal{I}} = \mathcal{L}(D, w_i^{(t)})$$

Dabei wird in der Regel für jedes Feature je ein Lerner für beide Polaritäten trainiert. Wenn also ein Lerner z.B. nur zu 40% korrekt klassifiziert, gibt es stets einen passenden Lerner, der exakt gegenteilige Vorhersagen macht und somit eine Präzision von 60% besitzt.

Ziel ist es, den besten Lerner in der Iteration auszuwählen und diesen dem Ensemble hinzuzufügen. Dazu werden die Gewichte der falsch vorhergesagten Datenpunkte summiert:

$$\varepsilon_j = \sum_{i=1}^{m} w_i^{(t)} \cdot I\left(y_i \neq h_j\left(\boldsymbol{x}_i\right)\right)$$

I bezeichnet die Indikatorfunktion, die 1 zurückgibt, wenn $y_i \neq h_j(\boldsymbol{x_i})$ erfüllt ist, und sonst 0. Das Modell mit dem geringsten Fehler ε_j wird in der Iteration als h_t mit Fehler ε_t ausgewählt.

Berechnung des Lernkoeffizienten

Der Koeffizient α_t für den ausgewählen schwachen Lerner h_t wird durch den minimalen Verlust berechnet. Daher wird dieser zunächst als exponentieller Verlust definiert:

$$L(h_t) = \sum_{i=1}^{m} w_i^{(t)} e^{-y_i h_t(x_i)}$$

Der Lernkoeffizient α_t wird nun als Faktor im Exponenten eingeführt:

$$L(h_t) = \sum_{i=1}^{m} w_i^{(t)} e^{-\alpha_t y_i h_t(x_i)}$$

Korrekt und falsch vorhergesagte Daten haben nun je folgenden Einfluss auf den Verlust:

$$\begin{aligned} y_i = h_t(x_i) &\implies y_i h_t(x_i) = 1 \\ &\rightsquigarrow \text{ Beitrag zum Verlust: } w_i^{(t)} e^{-\alpha_t} \\ y_i \neq h_t(x_i) &\implies y_i h_t(x_i) = -1 \\ &\rightsquigarrow \text{ Beitrag zum Verlust: } w_i^{(t)} e^{\alpha_t} \end{aligned}$$

So lässt sich der Verlust in je einen Teil für korrekt und falsch vorhergesagte Daten aufteilen:

$$L(h_t) = \sum_{y_i = h_t(x_i)} w_i^{(t)} e^{-\alpha_t} + \sum_{y_i \neq h_t(x_i)} w_i^{(t)} e^{\alpha_t}$$

Nun soll α_t so gewählt werden, dass der Fehler minimiert wird. Entprechend wird nach α_t abgeleitet und $L(h_t)'$=0 gesetzt:

$$\begin{aligned} \frac{dL(h_t)}{d\alpha_t} &= -e^{-\alpha_t} \sum_{y_i = h_t(x_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(x_i)} w_i^{(t)} = 0 \\ &\Leftrightarrow e^{2\alpha_t} = \frac{\sum_{y_i = h_t(x_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(x_i)} w_i^{(t)}} \\ &\Leftrightarrow \alpha_t = \frac{1}{2} \ln \left(\frac{\sum_{y_i = h_t(x_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(x_i)} w_i^{(t)}} \right) \end{aligned}$$

Da $\varepsilon_t = \sum_{h_t(x_i) \neq y_i} w_i^{(t)}$ ist und aus der Tatsache, dass die Summe der Gewichte 1 ist folgt, dass $1 - \varepsilon_t = \sum_{h_t(x_i) = y_i} w_i^{(t)}$ lässt sich α_t berechnen durch:

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

Dieser gibt an, wie stark die Vorhersage dieses schwachen Lerners im späteren Ensemble gewichtet wird und bestimmt die Aktualisierung der Gewichte für die nächste Iteration.

Aktualisierung der Gewichte

Die Gewichte der Trainingsdaten werden basierend auf dem zuvor bestimmten Koeffizienten wie folgt aktualisiert:

$$\begin{aligned} w_i^{(t+1)} &= w_i^{(t)} \cdot e^{-\alpha_t} \quad \text{für korrekt klassifizierte Datenpunke} \\ w_i^{(t+1)} &= w_i^{(t)} \cdot e^{\alpha_t} \quad \text{für falsch klassifizierte Datenpunke} \end{aligned}$$

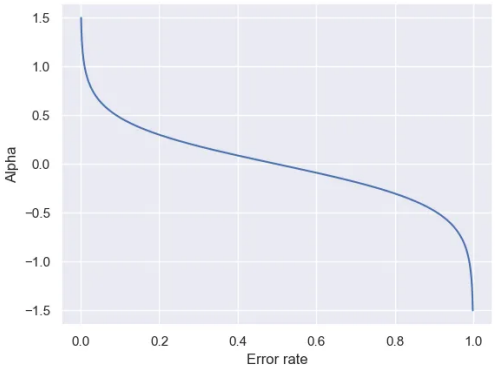


Abbildung 2: Verhalten von α_t in Anhängigkeit von ε_t

Dabei werden alle neuen Gewichte zuletzt normalisiert, damit ihre Summe nach der Aktualisierung wieder 1 ist.

$$Z_t = \sum_{j=1}^{m} w_i^{(t+1)} \qquad \text{(Normalisierungsfaktor)}$$

$$w_i^{(t+1)} = \frac{w_i^{(t+1)}}{Z_t}$$

Das Ergebnis des Algorithmus

Der Algorithmus gibt ein Gesamtmodell zurück, welches die Klassifizierung des Datenpunktes durch die gewichtete Summe aller schwachen Lerner darstellt:

$$H : X \to \{-1, +1\}$$

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

Data: Trainingsdatensatz D, Anzahl der Iterationen T.
Result: Finale Klassifikationsfunktion: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.

`// Initialisiere Gewichte`
$w_i^{(1)} = \frac{1}{m}$
for $t = 1$ **to** T **do**
  `// Trainiere schwache Lerner`
  $(h_j)_{j \in \mathcal{I}} \leftarrow \mathcal{L}(D, w_i^{(t)})$
  `// Berechne Fehler`
  for $j = 1$ **to** n **do**
    $\varepsilon_j = \sum_{i=1}^{m} w_i^{(t)} \cdot I(y_i \neq h_j(x_i))$
  Wähle Lerner h_j mit minimalem Fehler ε_j als h_t mit Fehler ε_t
  `// Berechne den Lernerkoeffizienten`
  $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
  `// Aktualisiere die Gewichte für die nächsten Iterationen`
  if $y_i = h_t(x_i)$ **then**
    $w_i^{(t+1)} \leftarrow w_i^{(t)} \cdot e^{-\alpha_t}$
  else
    $w_i^{(t+1)} \leftarrow w_i^{(t)} \cdot e^{\alpha_t}$
  `// Normalisiere Gewichte`
  $Z_t \leftarrow \sum_{j=1}^{m} w_i^{(t+1)}$
  for $i = 1$ **to** m **do**
    $w_i^{(t+1)} \leftarrow \frac{w_i^{(t+1)}}{Z_t}$

Output: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
`// Ende des Algorithmus`

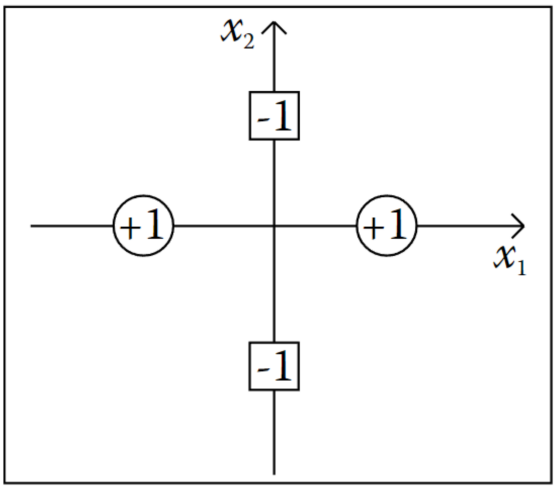


Abbildung 3: Visualisierung des XOR-Problems

Illustration des Algorithmus: das XOR-Problem

(Entnommen aus „The Top 10 Algorithms in Data Mining“ [WK09])

Ein Datensatz besteht aus nur 4 Datenpunken:

$$\left\{ \begin{array}{l} (x_1 = (0, +1), y_1 = +1) \\ (x_2 = (0, -1), y_2 = +1) \\ (x_3 = (+1, 0), y_3 = -1) \\ (x_4 = (-1, 0), y_4 = -1) \end{array} \right\}$$

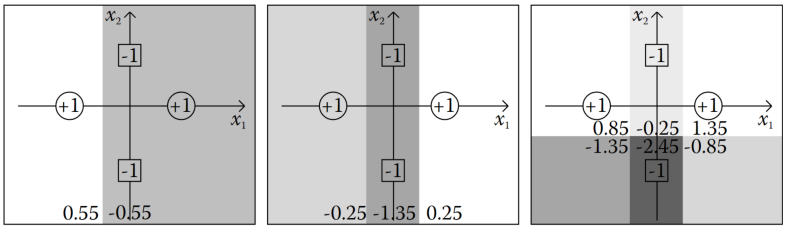
Wie aus Abbildung 3 leicht zu erkennen ist, kann keine einfache Trennlinie gezogen werden, um $+1$ und -1 voneinander zu trennen.
Nehmen wir nun an, dass durch den Lernalgorithmus nun acht Modelle als Funktionen vorliegen:

$$h_1(x) = \left\{ \begin{array}{ll} +1, & \text{wenn } (x_1 > -0.5) \\ -1, & \text{sonst} \end{array} \right. \quad h_2(x) = \left\{ \begin{array}{ll} -1, & \text{wenn } (x_1 > -0.5) \\ +1, & \text{sonst} \end{array} \right.$$

$$h_3(x) = \left\{ \begin{array}{ll} +1, & \text{wenn } (x_1 > +0.5) \\ -1, & \text{sonst} \end{array} \right. \quad h_4(x) = \left\{ \begin{array}{ll} -1, & \text{wenn } (x_1 > +0.5) \\ +1, & \text{sonst} \end{array} \right.$$

$$h_5(x) = \left\{ \begin{array}{ll} +1, & \text{wenn } (x_2 > -0.5) \\ -1, & \text{sonst} \end{array} \right. \quad h_6(x) = \left\{ \begin{array}{ll} -1, & \text{wenn } (x_2 > -0.5) \\ +1, & \text{sonst} \end{array} \right.$$

$$h_7(x) = \left\{ \begin{array}{ll} +1, & \text{wenn } (x_2 > +0.5) \\ -1, & \text{sonst} \end{array} \right. \quad h_8(x) = \left\{ \begin{array}{ll} -1, & \text{wenn } (x_2 > +0.5) \\ +1, & \text{sonst} \end{array} \right.$$



(b) 1st round (c) 2nd round (d) 3rd round

Abbildung 4: Visualisierung von AdaBoost auf dem XOR-Problem

1. Der erste Schritt besteht darin, den Basis-Lernalgorithmus auf den ursprünglichen Daten aufzurufen. h_2, h_3, h_5 und h_8 haben alle eine Klassifikationsfehler von 0.25. Angenommen, h_2 wird als erster Basis-Lerner ausgewählt. Ein Datensatz (x_1) wird falsch klassifiziert, daher beträgt der Fehler 0.25. Das Gewicht von h_2 beträgt ungefähr $\varepsilon_t \approx 0.55$. Abbildung 4(b) zeigt die Klassifikation und die Gewichtungen.

2. Das Gewicht von x_1 wird erhöht und der Basis-Lernalgorithmus erneut aufgerufen. Diesmal haben h_3, h_5 und h_8 gleiche Fehler. Angenommen, h_3 wird ausgewählt, dessen Gewicht 0.80 beträgt. Abbildung 4(c) zeigt die kombinierte Klassifikation von h_2 und h_3.

3. Das Gewicht von x_3 wird erhöht. Diesmal haben nur h_5 und h_8 die niedrigsten Fehler. Angenommen, h_5 wird ausgewählt, dessen Gewicht 1.10 beträgt. Abbildung 4(d) zeigt die kombinierte Klassifikation von h_2, h_3 und h_8. Durch die Kombination der unvollkommenen linearen Klassifikatoren hat AdaBoost einen nichtlinearen Klassifikator mit null Fehler erzeugt.

4 Praktische Anwendung und Beispiele

Vor allem wegen seiner Effizienz und Genauigkeit hat sich in einer Vielzahl von Anwendungen bewährt:

- **Bilderkennung und Computervision:** Gesichtserkennung [VJ01b]
- **Textklassifikation und Natural Language Processing**: Erkennung von Spam-Mail [PJM+22]
- **Medizinische Diagnostik:** Risiko/Erkennung von Krankheiten baserend auf Patientendaten [HGAA20]
- **Finanzwesen:** Vorhersage von Aktienkursbewegungen [ZLP16]

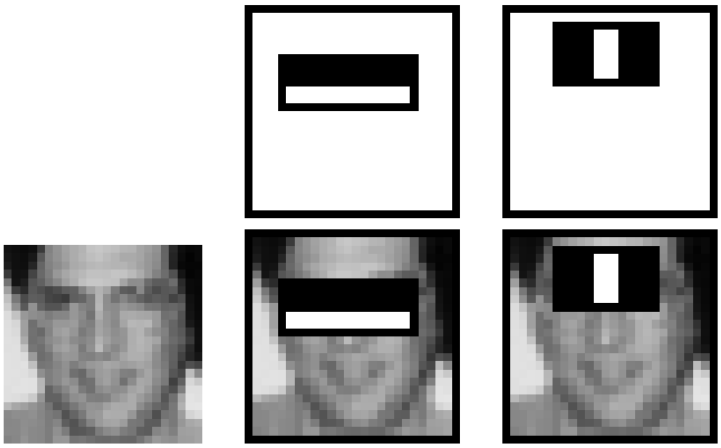


Abbildung 5: Anwendung von AdaBoost bei Computer Vision: Features messen Kontraste in bestimmten Gesichtsregionen[VJ01b]

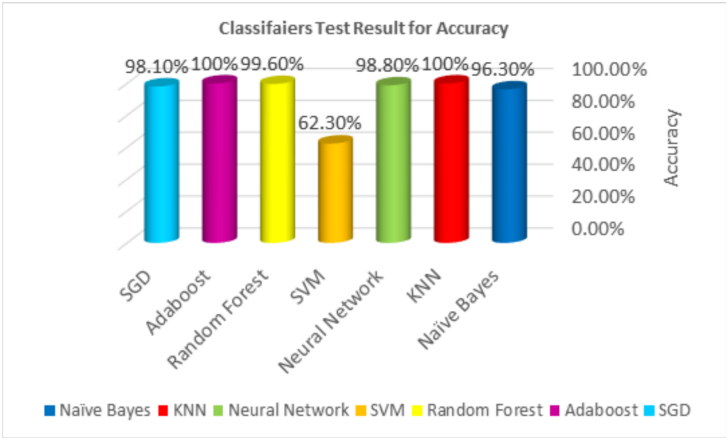


Abbildung 6: Erkennung von Spam-Mail durch AdaBoost im Vergleich zu anderen Verfahren[PJM+22]

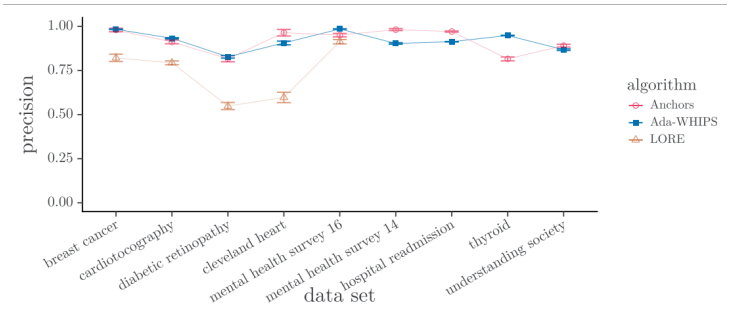


Abbildung 7: Diagnostik mit AdaBoost im Vergleich [HGAA20]

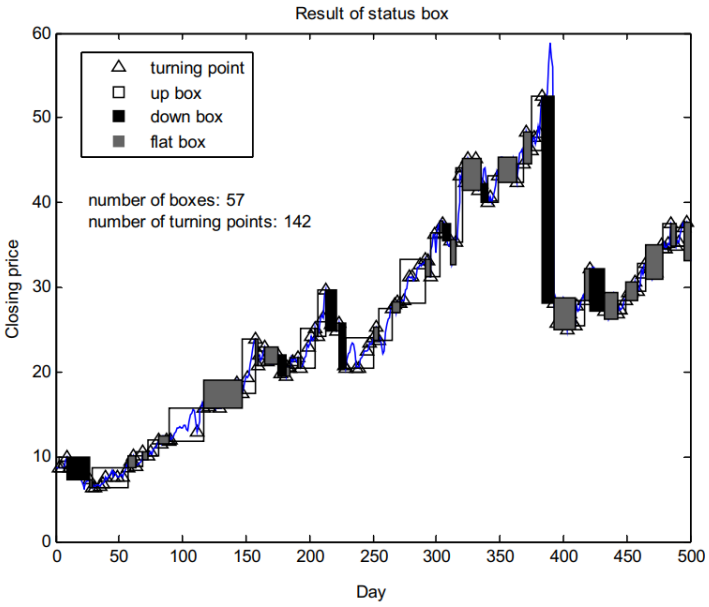


Abbildung 8: Vorhersage von Kursbewegungen mit AdaBoost. Boxen fassen Schlusspreise als Features zusammen. [ZLP16]

5 Vor- und Nachteile von AdaBoost

AdaBoost zeichnet sich unter anderem dadurch aus, dass der Algorithms benutzerfreundlich und flexibel ist, wodurch er leicht an spezifische Probleme angepasst werden kann. Ein weiterer Vorteil ist, dass AdaBoost automatisch wichtige Features identifiziert und diese entsprechend im Ensemble gewichtet. Letztlich neigt AdaBoost zudem weniger zum Overfitting, da nur schwache Lerner genutzt werden, deren Kapazität zu gering ist, um die Daten „auswendig“ zu lernen.

Jedoch ist AdaBoost anfällig für Ausreißer und verrauschte Daten, weil sich die adaptive Gewichtsanpassung mit jeder Iteration komulativ auf falsch klassifizierte Daten fokussiert und so versucht, schwer klassifizierbare Daten zu berücksichtigen. Zudem kann das Training auf großen Datensätzen sehr zeitintensiv sein, da oftmals für hunderte bis tausende Iterationen trainiert wird, wobei jedes mal eine große Menge von Lernern trainiert und evaluiert werden müssen. Zuletzt ist AdaBoost hauptsächlich für binäre Klassifikation ausgelegt und somit nicht für Regression einsetzbar und nur durch Erweiterungen für Multiklassen-Probleme genutzt werden kann.

6 Erweiterungen und Variationen von AdaBoost

AdaBoost, ursprünglich für binäre Klassifikation entwickelt, wurde durch verschiedene Erweiterungen für diverse Problemstellungen adaptiert.

- Variationen wie „AdaBoost.M1“ und „SAMME“ erweitern den Algorithmus für Multiklassen-Probleme. [HRZZ09]
- Kosten-sensitives AdaBoost passt Gewichtungen basierend auf bestimmten Fehlerarten an. [MSV10]
- Neben Entscheidungsstümpfen kann AdaBoost mit SVMs, Neuronalen Netzen oder jedem anderen Klassifikator kombiniert werden. [ZLP16]
- Robuste AdaBoost-Varianten minimieren die Auswirkung von Ausreißern. [VJ01a]
- Online AdaBoost aktualisiert Modelle mit sequenziellen Daten ohne Neutrainierung. [HGW$^+$13]
- Einige Varianten integrieren Feature-Auswahl direkt, um Interpretierbarkeit und Trainingseffizienz zu steigern. [WRM03]

Literatur

[HGAA20] Julian Hatwell, Mohamed Medhat Gaber, and R Muhammad Atif Azad. Ada-WHIPS: explaining AdaBoost classification with applications in the health sciences. *BMC Medical Informatics and Decision Making*, 20(1):1–25, 2020.

[HGW+13] Weiming Hu, Jun Gao, Yanguo Wang, Ou Wu, and Stephen Maybank. Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *IEEE Transactions on Cybernetics*, 44(1):66–82, 2013.

[HRZZ09] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[MSV10] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Cost-sensitive boosting. *IEEE Transactions on pattern analysis and machine intelligence*, 33(2):294–309, 2010.

[PJM+22] Manish Panwar, Jayesh Rajesh Jogi, Mahesh Vijay Mankar, Mohamed Alhassan, and Shreyas Kulkarni. Detection of Spam Email. *AJISE*, 1, 2022.

[VJ01a] Paul Viola and Michael Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. *Advances in neural information processing systems*, 14, 2001.

[VJ01b] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* IEEE, 2001.

[WK09] Xindong Wu and Vipin Kumar. *The Top Ten Algorithms in Data Mining.* CRC press, 2009.

[WRM03] Jianxin Wu, James M Rehg, and Matthew Mullin. Learning a rare event detection cascade by direct feature selection. *Advances in Neural Information Processing Systems*, 16, 2003.

[ZLP16] Xiao-dan Zhang, Ang Li, and Ran Pan. Stock trend prediction based on a new status box method and AdaBoost probabilistic support vector machine. *Applied Soft Computing*, 49:385–398, 2016.