Ausarbeitung zum Thema AdaBoost

Bachelor-Seminar „Top 10 Algorithms in Data Mining“

Marius Graf

12. November 2023

Inhaltsverzeichnis

Abstract: Diese Arbeit untersucht den AdaBoost-Algorithmus im Kontext des maschinellen Lernens. Sie beleuchtet seine Entstehung, Vorteile und Herausforderungen. Anhand von Anwendungsbeispielen aus verschiedenen Bereichen wird die Anwendbarkeit von AdaBoost dargestellt. Die Relevanz von AdaBoost in der aktuellen Datenwissenschaft wird abschließend erörtert.

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 schwierig zu klassifizierende Datenpunkte besser vorherzusagen.

2 Grundlagen des Boosting

Boosting ist eine Ensemble-Technik im Machine Learning, die mehrere sog. *„schwache Lerner“* kombiniert, um ein präzises 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 falsche Vorhersagen der vorherigen Modelle korrigiert, indem es die Gewichtung der Trainingsdaten anpasst. Das Ziel ist es, den systematischen Fehler (*Bias*) des Modells zu reduzieren und die Genauigkeit schrittweise zu verbessern, indem es sich auf schwierig zu klassifizierende Datenpunkte konzentriert.

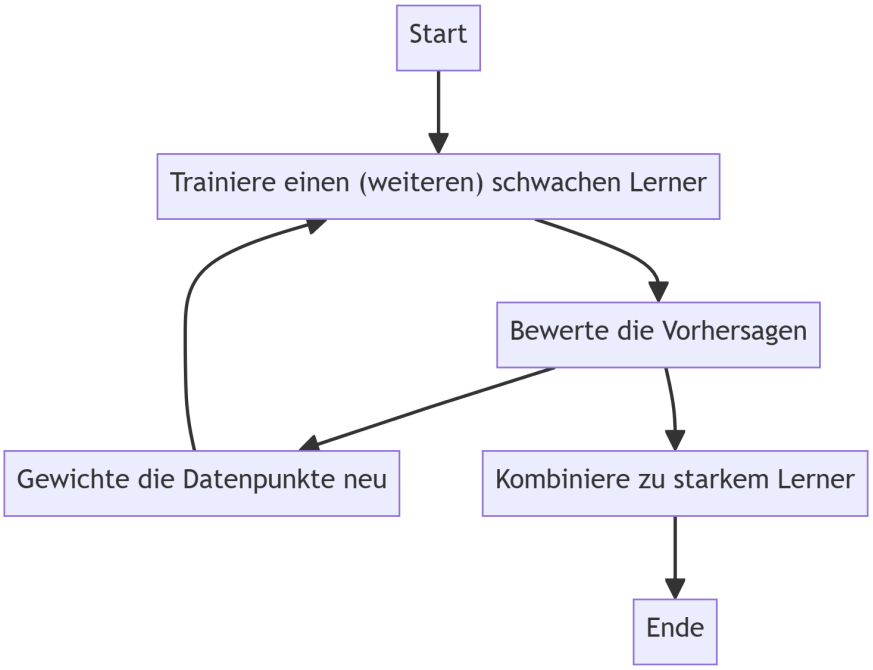


Abbildung 1: Flussdiagramm zur Veranschaulichung des Boosting-Konzepts

Beispiel:

Stellen wir uns vor, wir möchten ein Modell entwickeln, das den Preis von Häusern basierend auf verschiedenen Merkmalen wie Größe, Lage, Anzahl der Zimmer und Baujahr vorhersagt.

Haus	**Größe** $[m^2]$	**Lage**	**Preis**
Haus 1	100	Zentrum	500.000€
Haus 2	150	Vorort	300.000€
Haus 3	80	Zentrum	400.000€
Haus 4	120	Ländlich	200.000€

Tabelle 1: Beispielhafte Daten für Hauspreise basierend auf Größe und Lage

- Wir entscheiden uns zunächst für ein sehr einfaches Modell (schwacher Lerner), das den Preis nur anhand der Größe des Hauses vorhersagt. Dieses Modell geht davon aus, dass alle anderen Merkmale keinen Einfluss auf den Preis haben.
- In Wirklichkeit variieren die Hauspreise jedoch nicht nur aufgrund ihrer Größe, sondern auch aufgrund anderer Faktoren. Ein kleines Haus in einer begehrten Lage könnte teurer sein als ein großes Haus in einer weniger beliebten
- Da unser Modell nur die Größe berücksichtigt und alle anderen Faktoren ignoriert, wird es systematisch den Preis von Häusern in begehrten Lagen unterschätzen und den Preis von Häusern in weniger beliebten Gegenden überschätzen. Dieser systematische Fehler in den Vorhersagen ist der **Bias**.

In mehreren Iterationen wird beim Boosting nun das Gewicht der Datenpunkte so angepasst, dass sich das zweite Modell auf genau die Datenpunke fokussiert, welche zuvor falsch bzw. besonders schlecht vorhergesagt wurden. Somit ist hier eine deutlich größere Anpassung der Vorhersage zu erkennen als bei den Datenpunkten, die zuvor relativ gut vorhergasagt wurden.

Zuletzt werden alle schwachen Lerner zu einem *starken Lerner* als Ensemble zusammengefügt, wobei die Vorhersagen der einzelnen Modelle entsprechend ihrer individuellen Präzision gewichtet werden.

Haus	**Größe** $[m^2]$	**Lage**	**Preis**	**Vorhersage (It. 1)**	**Vorhersage (It. 2)**
Haus 1	100	Zentrum	500.000€	450.000€	490.000€
Haus 2	150	Vorort	300.000€	350.000€	310.000€
Haus 3	80	Zentrum	400.000€	380.000€	405.000€
Haus 4	120	Ländlich	200.000€	250.000€	210.000€

Tabelle 2: Beispielhafte Daten für Hauspreise und wie Boosting den Bias in mehreren Iterationen reduziert

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 maßgeblich geprägt. 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, wodurch er kontinuierlich seine Vorhersagegenauigkeit verbessert. Dieser einzigartige und adaptive Ansatz zur Fehlerkorrektur unterscheidet AdaBoost von anderen Boosting-Methoden und hat nicht nur die Effizienz von Boosting-Methoden für binäre Klassifikationsprobleme unter Beweis gestellt, sondern auch zu zahlreichen Weiterentwicklungen in diesem Bereich angeregt. [WK09]

Notation

- Sei $\mathcal{X}$ die Menge der Features mit $|\mathcal{X}| = n$ (Anzahl der Features) und $\mathcal{Y}$ 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 wird ein Lernalgorithmus $\mathcal{L}$ eine Hypothese bzw. einen Klassifizierer h zurück geben, der von $\mathcal{X}$ nach $\mathcal{Y}$ abbildet.
$$h : \mathcal{X} \to \mathcal{Y}, h(\boldsymbol{x}) = y$$

- T ist die Anzahl der gewünschten Trainingsiterationen.

- Bei jeder Iteration $t = 1, \ldots, T$ wird ein Datensatz $\mathcal{D}_t$ von D abgeleitet. Dabei wird jeder Datenpunkt mit einem Gewicht $\mathcal{D}_t(i)$ mit $i = 1, \ldots, m$ erweitert.

Initialisierung der Gewichte

Zu Beginn sind die Gewichte aller m Datenpunkte gleich verteilt:

$$\mathcal{D}_1(i) = \frac{1}{m}$$

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

$$\sum_{i=1}^{n} \mathcal{D}_t(i) = 1$$

Training der schwachen Lerner

Trainiere für $t = 1, \ldots, T$ Iterationen schwache Lerner unter berücksichtigung der aktuellen Gewichtung.

$$h_t = \mathcal{L}(D, \mathcal{D}_t)$$

(Hinweis: Abhängig vom konkreten Lernalgorithmus wird ggf. zusätzlich zum gewichteten Datensatz $\mathcal{D}_t$ auch der gesamte Datensatz D benötigt. Daher werden formal beide übegeben.)
Das Ziel ist es, den gewichteten Fehler zu minimieren:

$$\varepsilon_t = \sum_{i=1}^{n} D_t(i) I\left(y_i \neq h_t\left(\boldsymbol{x}_i\right)\right)$$

I ist die Indikatorfunktion, die 1 zurückgibt, wenn $y_i \neq h_t(\boldsymbol{x_i})$ erfüllt ist, und sonst 0. Das bedeutet, sie gibt 1 für jeden falsch vorhergesagten Datenpunkt zurück. ε_t stellt die gewichtete Summe aller falschen Vorhersagen des t-ten Modells dar. Das Modell mit dem geringsten Fehler wird in jeder Iteration ausgewählt. Ein Fehler von 1 zeigt, dass alle Vorhersagen falsch und 0 bedeutet, dass alle korrekt sind. Bei einem Fehler von 0.5 sind entsprechend die Hälfte der Vorhersagen richtig.

Wenn $\varepsilon_t > 0.5$, sind die Vorhersagen also schlechter als zufälliges Raten und der Algorithmus (in der ursprünglichen Version) wird gestoppt, da weitere Lerner das Modell nicht verbessern würden.

In der Praxis werden aber meist mehrere Modelle pro Iteration trainiert, wobei nicht bei einem Fehler von $\varepsilon_t > 0.5$ gestoppt wird. Schwache Lerner werden für jedes Feature trainiert und beide Polaritäten betrachtet (damit also insgesamt $2n$ Modelle). Wenn ein Modell z.B. nur zu 40% korrekt vorhersagt, wird durch Umkehrung der Polarität eine 60%ige Genauigkeit erreicht. So gibt es für jede Iteration stets eine Auswahl von Modellen, die als h_t gewählt werden können.

Berechnung des Lernkoeffizienten

Der Koeffizient α_t für den ausgewählen schwachen Lerner h_t wird wie folgt berechnet:

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

Dieser gibt an, wie stark die Vorhersage dieses schwachen Lerners im späteren Ensemble gewichtet wird.

Aktualisierung der Gewichte

Die Gewichte der Trainingsdaten werden basierend auf den Vorhersagen des zuvor bestimmten Koeffizienten aktualisiert:

$$\begin{aligned} \mathcal{D}_{t+1}(i) =& \mathcal{D}_t(i) \times e^{-\alpha_t} && \text{für korrekt klassifizierte Datenpunke} \\ \mathcal{D}_{t+1}(i) =& \mathcal{D}_t(i) \times e^{\alpha_t} && \text{für falsch klassifizierte Datenpunke} \end{aligned}$$

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

$$Z_t = \sum_{j=1}^{n} \mathcal{D}_{t+1}(j) \text{ (Normalisierungsfaktor)}$$

$$\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_{t+1}(i)}{Z_t}$$

Das Ergebnis des Algorithmus

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

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

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

Data: Trainingsdatensatz D, wobei $x_i \in \mathcal{X}$ und $y_i \in \{-1, 1\}$; Anzahl der Iterationen T.

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

```
// Initialisiere Gewichte
```
$\mathcal{D}_1(i) = \frac{1}{n}$

for $t = 1$ **to** T **do**

```
    // Trainiere einen schwachen Lerner unter Berücksichtigung der
       Gewichtung D_t(i)
```
    $h_t \leftarrow \mathcal{L}(D, \mathcal{D}_t)$

```
    // Berechne den gewichteten Fehler
```
    $\epsilon_t = \sum_{i=1}^{n} \mathcal{D}_t(i) I(y_i \neq h_t(x_i))$

    Wähle den Lerner mit dem geringsten Fehler als h_t

    if $\epsilon_t > 0.5$ **then**

        break

```
    // Berechne den Lernerkoeffizienten
```
    $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

```
    // Aktualisiere die Gewichte
```
    for $i = 1$ **to** n **do**

        if $y_i = h_t(x_i)$ **then**

            $\mathcal{D}_{t+1}(i) \leftarrow \mathcal{D}_t(i) \times e^{-\alpha_t}$

        else

            $\mathcal{D}_{t+1}(i) \leftarrow \mathcal{D}_t(i) \times e^{\alpha_t}$

```
    // Normalisiere Gewichte
```
    $Z_t \leftarrow \sum_{j=1}^{n} \mathcal{D}_{t+1}(j)$

    for $i = 1$ **to** n **do**

        $\mathcal{D}_{t+1}(i) \leftarrow \frac{\mathcal{D}_{t+1}(i)}{Z_t}$

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

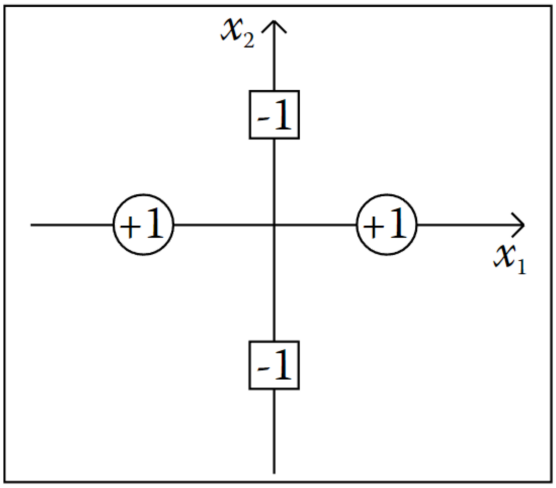


Abbildung 2: Visualisierung des XOR-Problems

Illustration des Algorithmus: das XOR-Problem

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

Ein Datensatz besteht aus nur 4 Datenpunken:

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

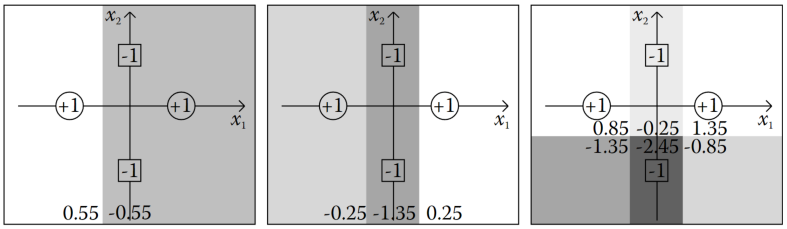
Wie aus Abbildung 2 leicht zu erkennen ist, kann keine einfache Trennlinie gezogen werden, um $+1$ und -1 voneinander zu trennen.
Nehmen wir nun an, dass durch den Lernalgorithmus nun acht Modelle als Funktionen vorliegen:

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

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

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

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



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

Abbildung 3: 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 3(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 3(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 3(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.

Siehe auch entsprechende Implementierung: [Gra23]

4 Praktische Anwendung und Beispiele

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

- **Bilderkennung und Computervision:** Gesichtserkennung [VJ01]
- **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]

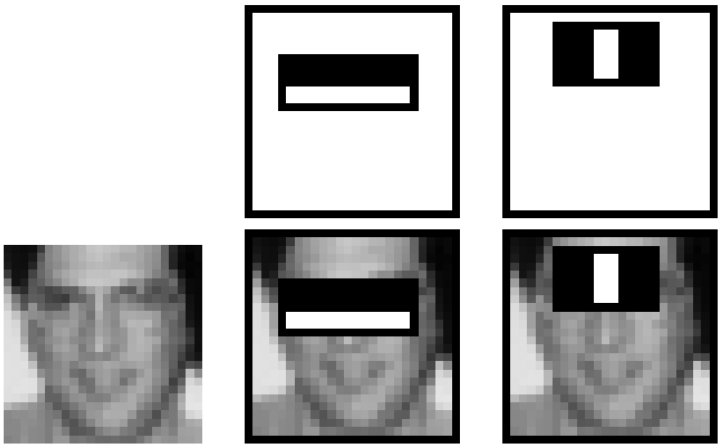


Abbildung 4: Anwendung von AdaBoost bei Computer Vision: Das erste Merkmal von AdaBoost misst den Intensitätsunterschied zwischen der Augenregion und den oberen Wangen, wobei die Augen oft dunkler sind. Das zweite Merkmal vergleicht die Intensität der Augen mit der Nasenbrücke.[VJ01]

5 Vor- und Nachteile von AdaBoost

AdaBoost ist einfach zu implementieren, vielseitig und benötigt oft keine Anpassung des Basislerners. Er ist weniger anfällig für Overfitting und kann wichtige Features automatisch identifizieren.

Nachteile sind seine Empfindlichkeit gegenüber verrauschten Daten und Außreißern, der potenzielle Zeitaufwand bei großen Datensätzen und seine Abhängigkeit vom Basislerner sowie seine Ausrichtung auf binäre Klassifikation.

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.
- Kosten-sensitives AdaBoost passt Gewichtungen basierend auf Fehlerkosten an.
- Neben Entscheidungsstümpfen kann AdaBoost mit SVMs oder Neuronalen Netzen kombiniert werden.
- Robuste AdaBoost-Varianten minimieren die Auswirkung von Ausreißern.
- Online AdaBoost aktualisiert Modelle mit sequenziellen Daten ohne Neutrainierung.

- Einige Varianten integrieren Feature-Auswahl direkt, um Interpretierbarkeit und Trainingseffizienz zu steigern.

7 Schlusswort

AdaBoost hat als Pionier des Ensemble-Lernens das maschinelle Lernen maßgeblich beeinflusst. Durch seine Einfachheit und Leistungsfähigkeit ist er ein zentrales Werkzeug für Datenwissenschaftler. Trotz neuer Techniken und Algorithmen bleibt AdaBoost ein Beispiel für die Verbindung von Theorie und Praxis. Diese Arbeit soll einen Einblick in AdaBoosts Vielseitigkeit geben und zukünftige Forschungen inspirieren.

Literatur

[Gra23] Marius Graf. Seminar zu "top ten algorithms in data mining", adaboost (ausarbeitung, präsentation, code). `https://github.com/Vinfeno/TTAIDM_AdaBoost`, 2023. Git repository.

[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.

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

[VJ01] 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.

[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.