Bergische Universität Wuppertal
Wissenschaftliches Rechnen und Hochleistungsrechnen
Dr. Marcel Schweitzer

Bachelor-Seminar „Top 10 Algorithms in Data Mining“

AdaBoost

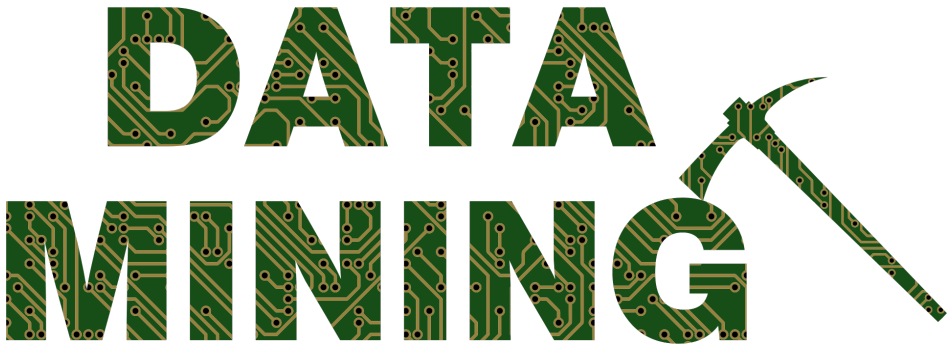
Marius Graf

06.12.2023

Inhalt

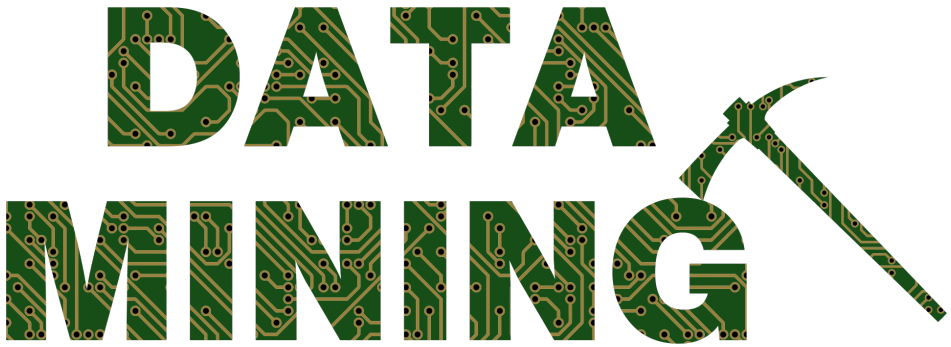
Inhalt

Was ist Data Mining?



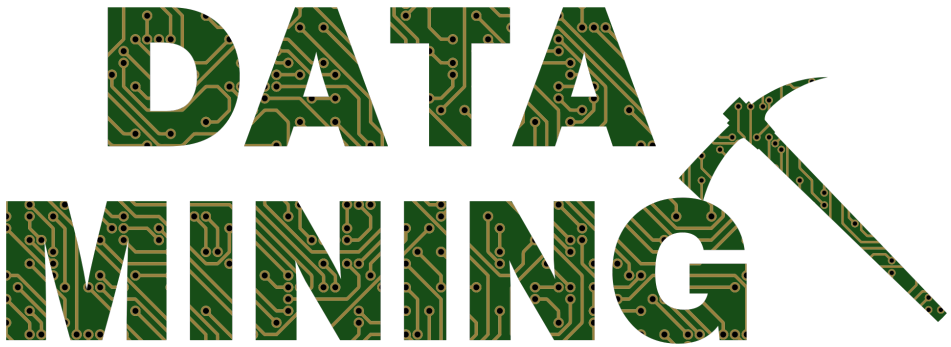
- Analysiert große Datenmengen, um Muster und Zusammenhänge zu erkennen.
- Nutzt dabei Methoden aus der Statistik, dem Machine Learning und Datenbanktechnologien.
- Spielt eine zentrale Rolle in der Forschung und Industrie, um Erkenntnisse zu gewinnen und Entscheidungen zu unterstützen.

Was ist Data Mining?



- Analysiert große Datenmengen, um Muster und Zusammenhänge zu erkennen.
- Nutzt dabei Methoden aus der Statistik, dem Machine Learning und Datenbanktechnologien.
- Spielt eine zentrale Rolle in der Forschung und Industrie, um Erkenntnisse zu gewinnen und Entscheidungen zu unterstützen.

Was ist Data Mining?



- Analysiert große Datenmengen, um Muster und Zusammenhänge zu erkennen.
- Nutzt dabei Methoden aus der Statistik, dem Machine Learning und Datenbanktechnologien.
- Spielt eine zentrale Rolle in der Forschung und Industrie, um Erkenntnisse zu gewinnen und Entscheidungen zu unterstützen.

Was sind Ensemble-Methoden?

- **Ensemble-Verfahren:** Kombinieren mehrere Modelle für präzisere Vorhersagen
- **Fehlerminimierung:** Reduzieren von systematische Fehlern in Modellprognosen
- **Arten von Ensemble-Methoden:**
  - Bagging
  - Stacking
  - Boosting
- **AdaBoost gehört zu den Boosting-Verfahren**

Was sind Ensemble-Methoden?

- **Ensemble-Verfahren:** Kombinieren mehrere Modelle für präzisere Vorhersagen
- **Fehlerminimierung:** Reduzieren von systematische Fehlern in Modellprognosen
- **Arten von Ensemble-Methoden:**
  - Bagging
  - Stacking
  - Boosting
- **AdaBoost gehört zu den Boosting-Verfahren**

Was sind Ensemble-Methoden?

- **Ensemble-Verfahren:** Kombinieren mehrere Modelle für präzisere Vorhersagen
- **Fehlerminimierung:** Reduzieren von systematische Fehlern in Modellprognosen
- **Arten von Ensemble-Methoden:**
  - Bagging
  - Stacking
  - Boosting
- **AdaBoost gehört zu den Boosting-Verfahren**

Was sind Ensemble-Methoden?

- **Ensemble-Verfahren:** Kombinieren mehrere Modelle für präzisere Vorhersagen
- **Fehlerminimierung:** Reduzieren von systematische Fehlern in Modellprognosen
- **Arten von Ensemble-Methoden:**
  - Bagging
  - Stacking
  - Boosting
- **AdaBoost gehört zu den Boosting-Verfahren**

Inhalt

Die Grundidee

- Boosting ist eine Ensemble-Methode, die schwache Lerner kombiniert, um ein starkes Gesamtmodell zu bilden.
- Ein schwacher Lerner ist ein Modell, das nur minimal besser als Zufall vorhersagt, indem es schwache Zusammenhänge in den Daten erkennt.
- Durch iterative Anpassung der Gewichtung von Trainingsdaten korrigiert jedes neue Modell die Fehler seiner Vorgänger.
- Das Verfahren zielt darauf ab, den Bias zu verringern und die Vorhersagegenauigkeit für schwer klassifizierbare Beispiele zu erhöhen.

Die Grundidee

- Boosting ist eine Ensemble-Methode, die schwache Lerner kombiniert, um ein starkes Gesamtmodell zu bilden.
- Ein schwacher Lerner ist ein Modell, das nur minimal besser als Zufall vorhersagt, indem es schwache Zusammenhänge in den Daten erkennt.
- Durch iterative Anpassung der Gewichtung von Trainingsdaten korrigiert jedes neue Modell die Fehler seiner Vorgänger.
- Das Verfahren zielt darauf ab, den Bias zu verringern und die Vorhersagegenauigkeit für schwer klassifizierbare Beispiele zu erhöhen.

Die Grundidee

- Boosting ist eine Ensemble-Methode, die schwache Lerner kombiniert, um ein starkes Gesamtmodell zu bilden.
- Ein schwacher Lerner ist ein Modell, das nur minimal besser als Zufall vorhersagt, indem es schwache Zusammenhänge in den Daten erkennt.
- Durch iterative Anpassung der Gewichtung von Trainingsdaten korrigiert jedes neue Modell die Fehler seiner Vorgänger.
- Das Verfahren zielt darauf ab, den Bias zu verringern und die Vorhersagegenauigkeit für schwer klassifizierbare Beispiele zu erhöhen.

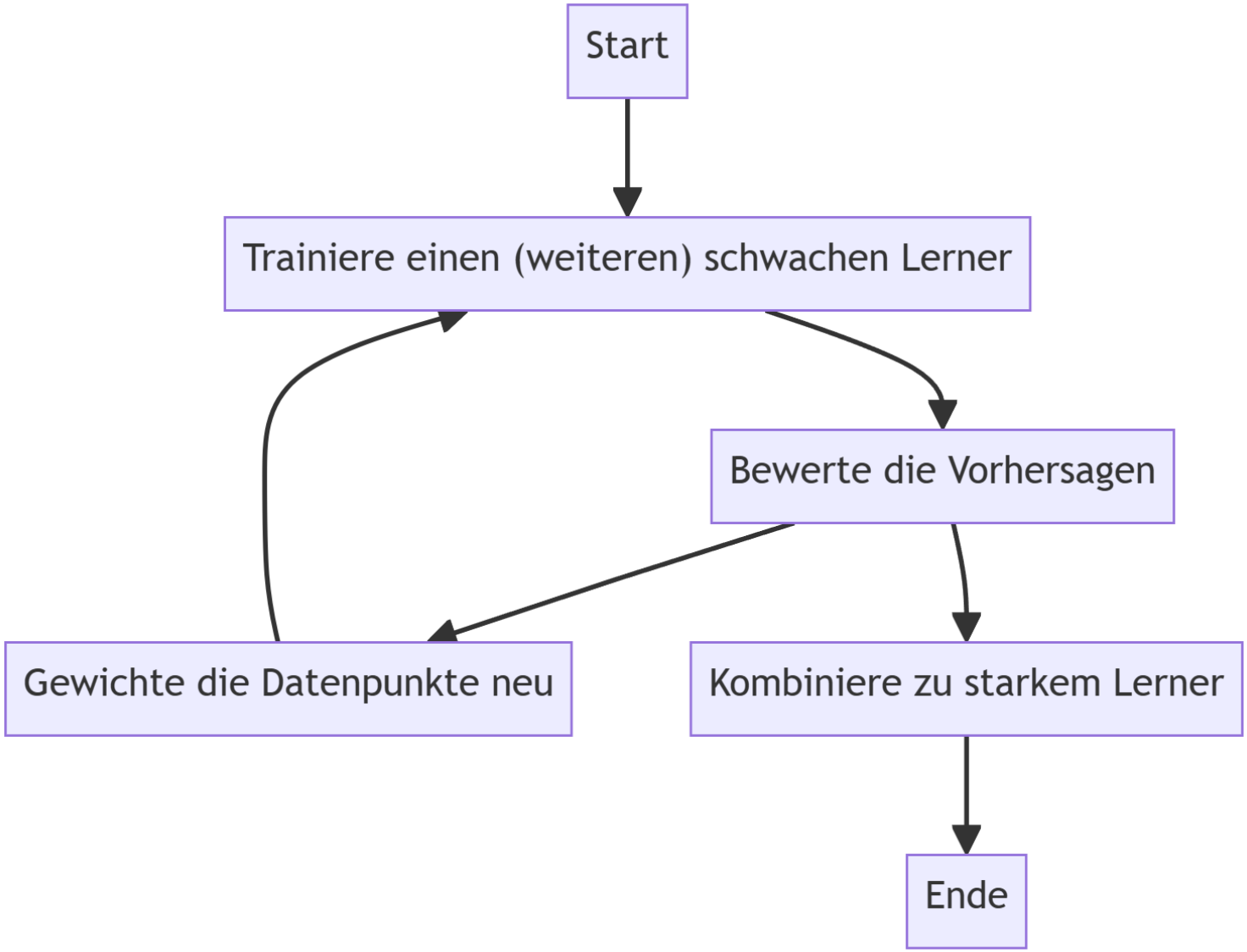
Die Grundidee

- Boosting ist eine Ensemble-Methode, die schwache Lerner kombiniert, um ein starkes Gesamtmodell zu bilden.
- Ein schwacher Lerner ist ein Modell, das nur minimal besser als Zufall vorhersagt, indem es schwache Zusammenhänge in den Daten erkennt.
- Durch iterative Anpassung der Gewichtung von Trainingsdaten korrigiert jedes neue Modell die Fehler seiner Vorgänger.
- Das Verfahren zielt darauf ab, den Bias zu verringern und die Vorhersagegenauigkeit für schwer klassifizierbare Beispiele zu erhöhen.

Die Grundidee

- Boosting ist eine Ensemble-Methode, die schwache Lerner kombiniert, um ein starkes Gesamtmodell zu bilden.
- Ein schwacher Lerner ist ein Modell, das nur minimal besser als Zufall vorhersagt, indem es schwache Zusammenhänge in den Daten erkennt.
- Durch iterative Anpassung der Gewichtung von Trainingsdaten korrigiert jedes neue Modell die Fehler seiner Vorgänger.
- Das Verfahren zielt darauf ab, den Bias zu verringern und die Vorhersagegenauigkeit für schwer klassifizierbare Beispiele zu erhöhen.

Veranschaulichung



Beispiel

Vorhersage von Hauspreisen

- 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: Beispielhafte Daten für Hauspreise basierend auf Größe und Lage

Beispiel

Vorhersage von Hauspreisen

- Zunächst wählen wir ein sehr einfaches Modell (schwacher Lerner), das den Preis nur anhand der Größe des Hauses vorhersagt.
- In Wirklichkeit variieren die Hauspreise jedoch nicht nur aufgrund ihrer Größe, sondern auch aufgrund anderer Faktoren. Gegend.
- 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**.

Beispiel

Vorhersage von Hauspreisen

- Zunächst wählen wir ein sehr einfaches Modell (schwacher Lerner), das den Preis nur anhand der Größe des Hauses vorhersagt.
- In Wirklichkeit variieren die Hauspreise jedoch nicht nur aufgrund ihrer Größe, sondern auch aufgrund anderer Faktoren. Gegend.
- 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**.

Beispiel

Vorhersage von Hauspreisen

- Zunächst wählen wir ein sehr einfaches Modell (schwacher Lerner), das den Preis nur anhand der Größe des Hauses vorhersagt.
- In Wirklichkeit variieren die Hauspreise jedoch nicht nur aufgrund ihrer Größe, sondern auch aufgrund anderer Faktoren. Gegend.
- 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**.

Beispiel

Vorhersage von Hauspreisen

- Beim Boosting wird das Gewicht von Datenpunkten iterativ so angepasst, dass sich nachfolgende Modelle verstärkt auf zuvor schlecht vorhergesagte Fälle konzentrieren.

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

Inhalt

Einführung

- „Adaptive Boosting“
- AdaBoost, entwickelt in den 1990ern von Freund und Schapire, ist ein einflussreiches Verfahren für binäre Klassifikation im maschinellen Lernen.
- Es nutzt eine iterative Boosting-Methode, die die Gewichtung falsch klassifizierter Datenpunkte erhöht, um die Vorhersagegenauigkeit zu verbessern.
- AdaBoosts spezielle adaptive Fehlerkorrektur hebt es von anderen Boosting-Methoden ab und hat zu vielen Weiterentwicklungen geführt.

[WK09]

Einführung

- „Adaptive Boosting“
- AdaBoost, entwickelt in den 1990ern von Freund und Schapire, ist ein einflussreiches Verfahren für binäre Klassifikation im maschinellen Lernen.
- Es nutzt eine iterative Boosting-Methode, die die Gewichtung falsch klassifizierter Datenpunkte erhöht, um die Vorhersagegenauigkeit zu verbessern.
- AdaBoosts spezielle adaptive Fehlerkorrektur hebt es von anderen Boosting-Methoden ab und hat zu vielen Weiterentwicklungen geführt.

[WK09]

Einführung

- „Adaptive Boosting“
- AdaBoost, entwickelt in den 1990ern von Freund und Schapire, ist ein einflussreiches Verfahren für binäre Klassifikation im maschinellen Lernen.
- Es nutzt eine iterative Boosting-Methode, die die Gewichtung falsch klassifizierter Datenpunkte erhöht, um die Vorhersagegenauigkeit zu verbessern.
- AdaBoosts spezielle adaptive Fehlerkorrektur hebt es von anderen Boosting-Methoden ab und hat zu vielen Weiterentwicklungen geführt.

[WK09]

Einführung

- „Adaptive Boosting“
- AdaBoost, entwickelt in den 1990ern von Freund und Schapire, ist ein einflussreiches Verfahren für binäre Klassifikation im maschinellen Lernen.
- Es nutzt eine iterative Boosting-Methode, die die Gewichtung falsch klassifizierter Datenpunkte erhöht, um die Vorhersagegenauigkeit zu verbessern.
- AdaBoosts spezielle adaptive Fehlerkorrektur hebt es von anderen Boosting-Methoden ab und hat zu vielen Weiterentwicklungen geführt.

[WK09]

Vereinfachte Sicht auf den Algorithmus

Input: Datensatz, Lernalgorithmus

1 Initialisiere Gewichte des Datensatzes
2 **for** $t = 1$ **to** T **do**
3 Trainiere schwache Lerner mit gewichtetem Datensatz
4 Bestimme Fehler der Lerner
5 Wähle schwachen Lerner mit geringstem Fehler
6 Berechne Lernkoeffizienten
7 Gewichte Datenpunkte neu

Output: Starker Lerner (Ensemble)

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 gleich verteilt

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

- Die Summe der Gewichte ist stets 1

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

Initialisierung der Gewichte

- Zu beginn sind die Gewichte gleich verteilt

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

- Die Summe der Gewichte ist 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.

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

- Das Ziel ist es, den gewichteten Fehler zu minimieren:

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

wähle daher den Lerner mit dem geringsten Fehler.

- Indikatorfunktion

$$I(A) = \begin{cases} 1, & \text{wenn } A \\ 0, & \text{sonst} \end{cases}$$

Training der schwachen Lerner

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

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

- Das Ziel ist es, den gewichteten Fehler zu minimieren:

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

wähle daher den Lerner mit dem geringsten Fehler.

- Indikatorfunktion

$$I(A) = \begin{cases} 1, & \text{wenn } A \\ 0, & \text{sonst} \end{cases}$$

Training der schwachen Lerner

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

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

- Das Ziel ist es, den gewichteten Fehler zu minimieren:

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

wähle daher den Lerner mit dem geringsten Fehler.

- Indikatorfunktion

$$I(A) = \begin{cases} 1, & \text{wenn } A \\ 0, & \text{sonst} \end{cases}$$

Berechnung des Lernkoeffizienten

- Zu dem ausgewählten Lerner h_t wird nun ein **Lernkoeffizient** α_t berechnet:

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

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

Berechnung des Lernkoeffizienten

- Zu dem ausgewählten Lerner h_t wird nun ein **Lernkoeffizient** α_t berechnet:

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

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

Aktualisierung der Gewichte

- Die neuen Gewichte der Daten für den nächsten Durchlauf werden berechnet durch

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

- Die neuen Gewichte müssen anschließend normalisiert werden, damit ihre Summe wieder 1 ist:

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

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

Aktualisierung der Gewichte

- Die neuen Gewichte der Daten für den nächsten Durchlauf werden berechnet durch

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

- Die neuen Gewichte müssen anschließend normalisiert werden, damit ihre Summe 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:

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

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

Der vollständige Algorithmus I

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
1  D_1(i) = 1/n
2  for t = 1 to T do
       // Trainiere einen schwachen Lerner unter Berücksichtigung der
          Gewichtung D_t(i)
3      h_t ← L(D, D_t)
       // Berechne den gewichteten Fehler
4      ε_t = Σ_{i=1}^{n} D_t(i) I(y_i ≠ h_t(x_i))
5      Wähle den Lerner mit dem geringsten Fehler als h_t
6      if ε_t > 0.5 then
7          break
8
       // Berechne den Lernerkoeffizienten
9      α_t = 1/2 ln((1 − ε_t)/ε_t)
       // Algorithmus wird fortgesetzt...
```

Der vollständige Algorithmus II

```
// Fortsetzung des Algorithmus
// Weiterhin innerhalb des For-Loops
// Aktualisiere die Gewichte für die nächsten Iterationen
```

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

```
// Ende des Algorithmus
```

Beispiel und Illustration

Das XOR-Problem

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

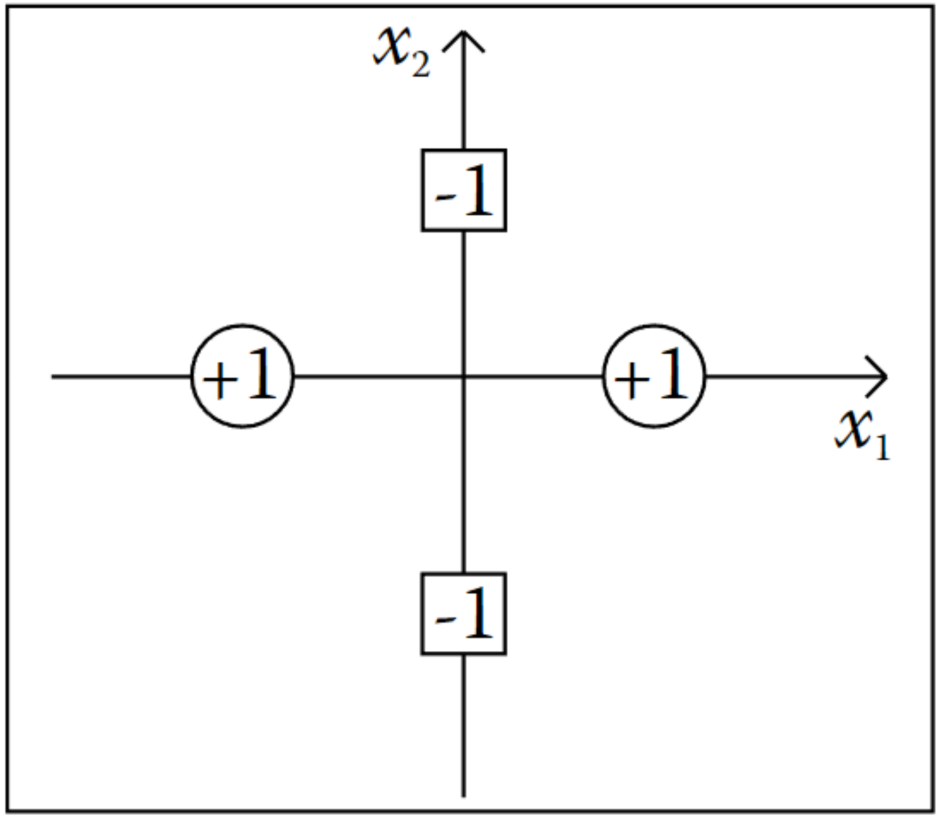
$$h_1(x) = \left\{ \begin{array}{ll} +1, & \text{wenn } (x_1 > -0.5) \\ -1, & \text{sonst} \end{array} \right. \quad h_2(x) = \left\{ \begin{array}{ll} -1, & \text{wenn } (x_1 > -0.5) \\ +1, & \text{sonst} \end{array} \right.$$

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

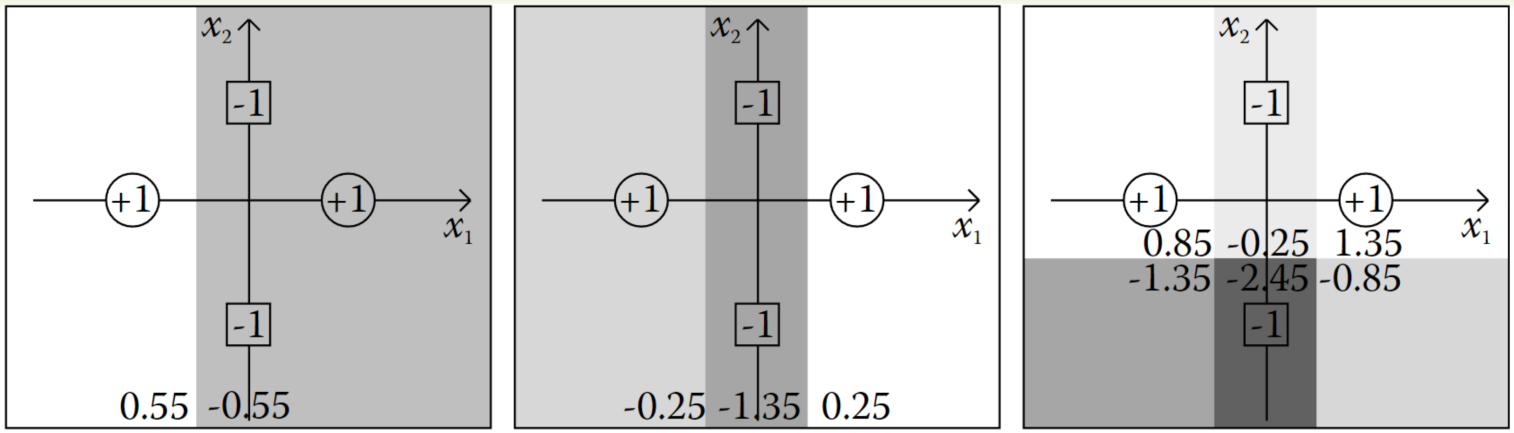
$$h_5(x) = \left\{ \begin{array}{ll} +1, & \text{wenn } (x_2 > -0.5) \\ -1, & \text{sonst} \end{array} \right. \quad h_6(x) = \left\{ \begin{array}{ll} -1, & \text{wenn } (x_2 > -0.5) \\ +1, & \text{sonst} \end{array} \right.$$

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

Das XOR-Problem



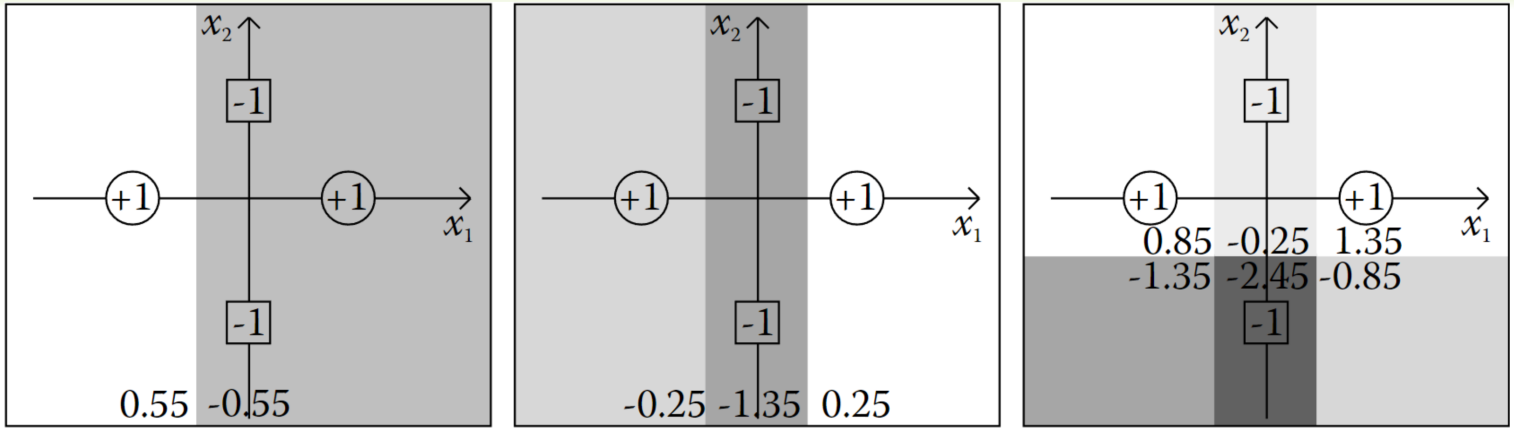
Das XOR-Problem



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

1. Basis-Lernalgorithmus wird mit ursprünglichen Daten trainiert; h_2 wird ausgewählt mit einem Fehler von 0.25 und einem Gewicht von ca. 0.55.
2. Nach Erhöhung des Gewichts von x_1 wird h_3 mit einem Fehler von 0.25 und einem Gewicht von 0.80 ausgewählt.
3. Gewicht von x_3 steigt, h_5 wird mit einem Gewicht von 1.10 ausgewählt, was zu einem nichtlinearen Klassifikator ohne Fehler führt.

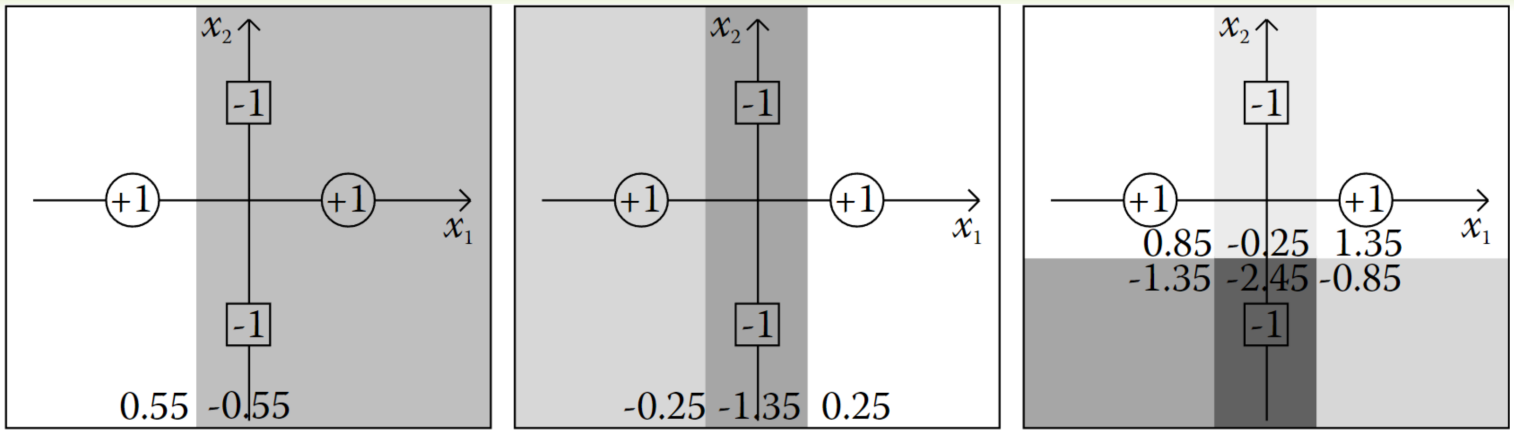
Das XOR-Problem



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

1. Basis-Lernalgorithmus wird mit ursprünglichen Daten trainiert; h_2 wird ausgewählt mit einem Fehler von 0.25 und einem Gewicht von ca. 0.55.
2. Nach Erhöhung des Gewichts von x_1 wird h_3 mit einem Fehler von 0.25 und einem Gewicht von 0.80 ausgewählt.
3. Gewicht von x_3 steigt, h_5 wird mit einem Gewicht von 1.10 ausgewählt, was zu einem nichtlinearen Klassifikator ohne Fehler führt.

Das XOR-Problem



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

1. Basis-Lernalgorithmus wird mit ursprünglichen Daten trainiert; h_2 wird ausgewählt mit einem Fehler von 0.25 und einem Gewicht von ca. 0.55.
2. Nach Erhöhung des Gewichts von x_1 wird h_3 mit einem Fehler von 0.25 und einem Gewicht von 0.80 ausgewählt.
3. Gewicht von x_3 steigt, h_5 wird mit einem Gewicht von 1.10 ausgewählt, was zu einem nichtlinearen Klassifikator ohne Fehler führt.

Inhalt

Praktische Anwendung und Beispiele

- **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]

Praktische Anwendung und Beispiele

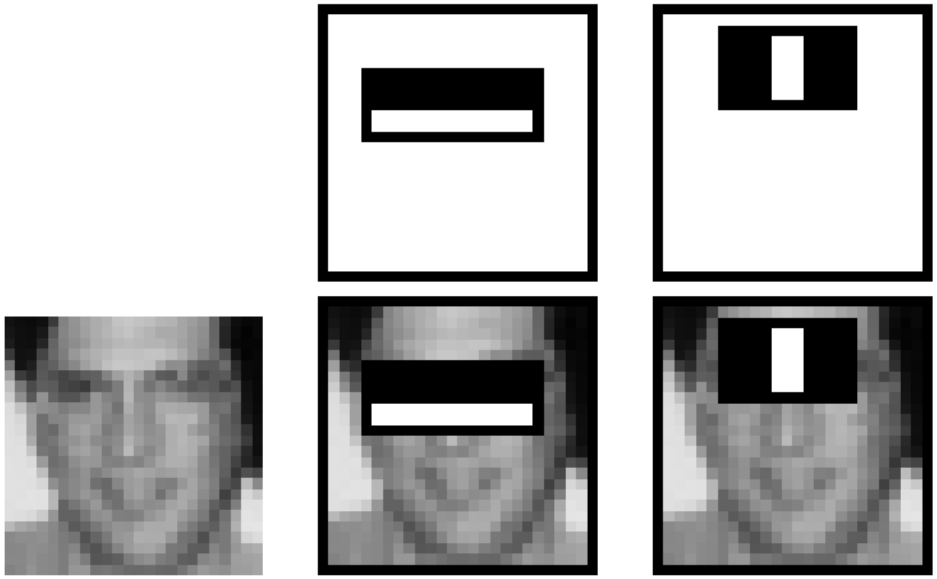


Abbildung: Anwendung von AdaBoost bei Computer Vision: Das erste Merkmal von AdaBoost misst den Intensitätsunterschied zwischen der Augenregion und den oberen Wangen, wobei die Augen oft dunkler sind. Das zweite Merkmal vergleicht die Intensität der Augen mit der Nasenbrücke.[VJ01]

Inhalt

Vor- und Nachteile von AdaBoost

- AdaBoost ist benutzerfreundlich, flexibel, und identifiziert automatisch wichtige Features, wobei es weniger zu Overfitting neigt.
- Es ist anfällig für verrauschte Daten und Außreißer, kann bei großen Datensätzen zeitintensiv sein und ist hauptsächlich für binäre Klassifikation ausgelegt.

Vor- und Nachteile von AdaBoost

- AdaBoost ist benutzerfreundlich, flexibel, und identifiziert automatisch wichtige Features, wobei es weniger zu Overfitting neigt.
- Es ist anfällig für verrauschte Daten und Außreißer, kann bei großen Datensätzen zeitintensiv sein und ist hauptsächlich für binäre Klassifikation ausgelegt.

Inhalt

Erweiterungen und Variationen von AdaBoost

- ursprünglich für binäre Klassifikation entwickelt, 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.

Erweiterungen und Variationen von AdaBoost

- ursprünglich für binäre Klassifikation entwickelt, 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.

Erweiterungen und Variationen von AdaBoost

- ursprünglich für binäre Klassifikation entwickelt, 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.

Erweiterungen und Variationen von AdaBoost

- ursprünglich für binäre Klassifikation entwickelt, 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.

Erweiterungen und Variationen von AdaBoost

- 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.

Erweiterungen und Variationen von AdaBoost

- 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.

Erweiterungen und Variationen von AdaBoost

- 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.

Inhalt

Literatur I

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.

Manish Panwar, Jayesh Rajesh Jogi, Mahesh Vijay Mankar, Mohamed Alhassan, and Shreyas Kulkarni.
Detection of spam email.
AJISE, 1, 2022.

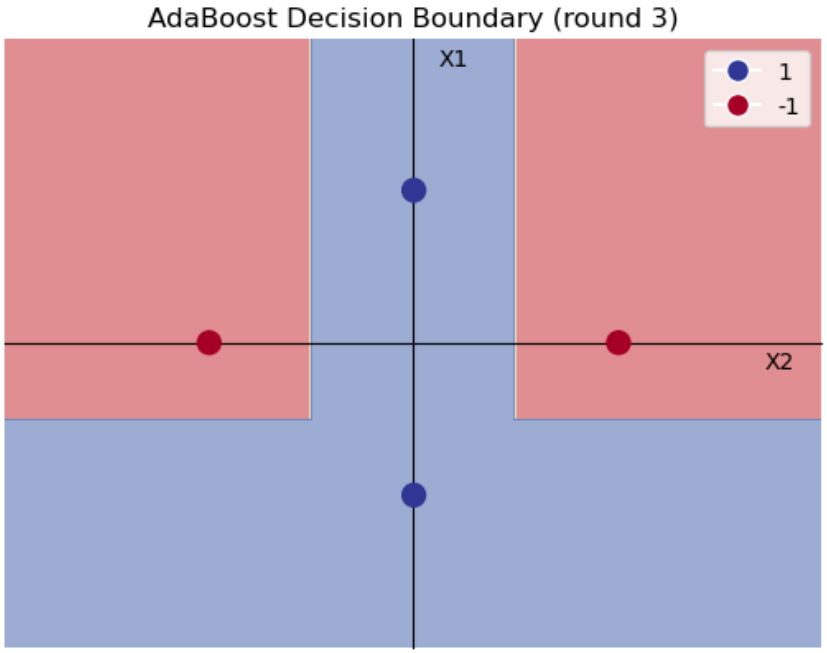
Literatur II

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.

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

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.

Zusatzmaterial



Umsetzungen und Beispiele von AdaBoost + diese Präsentation mit Ausarbeitung in LaTeXauf *GitHub*.

Danke

Vielen Dank für die Aufmerksamkeit!