



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

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial

# Inhalt

- 1 **Einleitung**
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial

# 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 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 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 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 **systematischen 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 **systematischen 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 **systematischen 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 **systematischen Fehlern** in Modellprognosen
- ▶ **Arten von Ensemble-Methoden:**
  - ▶ Bagging
  - ▶ Stacking
  - ▶ Boosting
- ▶ **AdaBoost gehört zu den Boosting-Verfahren**

# Inhalt

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial

# Die Grundidee

- ▶ Boosting kombiniert **schwache Lerner** zu einem **starken Gesamtmodell**
- ▶ **Schwacher Lerner**: Modell, das nur **geringfügig besser** ist als **zufälliges Raten**
- ▶ **Passe Gewichtung** der Trainingsdaten iterativ **an**, damit neue Modelle die **Fehler der Vorgänger korrigieren**
- ▶  $\rightsquigarrow$  verringerter **Bias, bessere Vorhersagegenauigkeit** für schwer klassifizierbare Beispiele

# Die Grundidee

- ▶ Boosting kombiniert **schwache Lerner** zu einem **starken Gesamtmodell**
- ▶ **Schwacher Lerner:** Modell, das nur **geringfügig besser** ist als **zufälliges Raten**
- ▶ **Passe Gewichtung** der Trainingsdaten iterativ **an**, damit neue Modelle die **Fehler der Vorgänger korrigieren**
- ▶  $\rightsquigarrow$  verringerter **Bias, bessere Vorhersagegenauigkeit** für schwer klassifizierbare Beispiele

# Die Grundidee

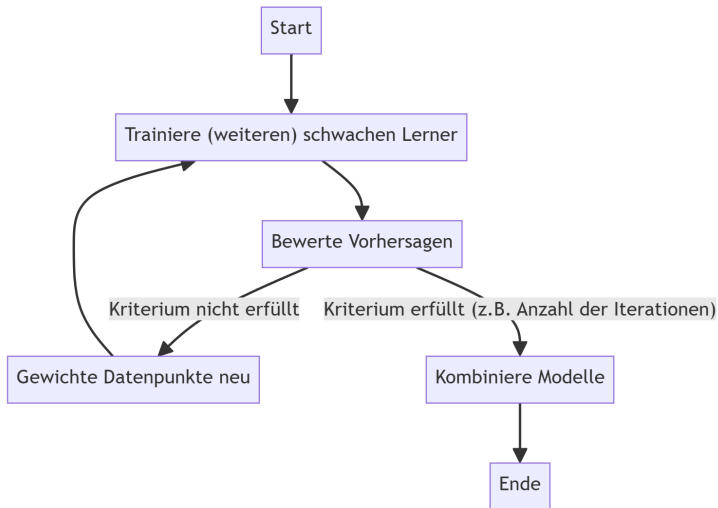
- ▶ Boosting kombiniert **schwache Lerner** zu einem **starken Gesamtmodell**
- ▶ **Schwacher Lerner:** Modell, das nur **geringfügig besser** ist als **zufälliges Raten**
- ▶ **Passe Gewichtung** der Trainingsdaten iterativ **an**, damit neue Modelle die **Fehler der Vorgänger korrigieren**
- ▶  $\rightsquigarrow$  verringerter **Bias, bessere Vorhersagegenauigkeit** für schwer klassifizierbare Beispiele



# Die Grundidee

- ▶ Boosting kombiniert **schwache Lerner** zu einem **starken Gesamtmodell**
- ▶ **Schwacher Lerner:** Modell, das nur **geringfügig besser** ist als **zufälliges Raten**
- ▶ **Passe Gewichtung** der Trainingsdaten iterativ **an**, damit neue Modelle die **Fehler der Vorgänger korrigieren**
- ▶  $\rightsquigarrow$  verringerter **Bias, bessere Vorhersagegenauigkeit** für schwer klassifizierbare Beispiele

# 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	440.000€
Haus 2	150	Vorort	500.000€
Haus 3	80	Zentrum	400.000€
Haus 4	120	Ländlich	200.000€

# Beispiel

## Vorhersage von Hauspreisen

- ▶ **Einfaches Modell** (schwacher Lerner): Preisvorhersage nur anhand von **Größe**
- ▶ Tatsächlich spielen auch **andere Faktoren** (z.B. Lage) eine Rolle
- ▶  $\rightsquigarrow$  **Bias** des schwachen Lernalgorithmus:
  - ▶ Preis von Häusern in guter Lage wird **unterschätzt**
  - ▶ Preis von Häusern in schlechter Lage wird **überschätzt**

# Beispiel

## Vorhersage von Hauspreisen

- ▶ **Einfaches Modell** (schwacher Lerner): Preisvorhersage nur anhand von **Größe**
- ▶ Tatsächlich spielen auch **andere Faktoren** (z.B. Lage) eine Rolle
- ▶  $\rightsquigarrow$  **Bias** des schwachen Lernalgorithmus:
  - ▶ Preis von Häusern in guter Lage wird **unterschätzt**
  - ▶ Preis von Häusern in schlechter Lage wird **überschätzt**

# Beispiel

## Vorhersage von Hauspreisen

- ▶ **Einfaches Modell** (schwacher Lerner): Preisvorhersage nur anhand von **Größe**
- ▶ Tatsächlich spielen auch **andere Faktoren** (z.B. Lage) eine Rolle
- ▶  $\rightsquigarrow$  **Bias** des schwachen Lernalgorithmus:
  - ▶ Preis von Häusern in guter Lage wird **unterschätzt**
  - ▶ Preis von Häusern in schlechter Lage wird **überschätzt**

# Beispiel

## Vorhersage von Hauspreisen

- **Boosting:** Passe iterativ Gewicht der Datenpunkte so an, dass nächstes Modell verstärkt die schlecht vorhergesagten Fälle beachtet

$P$  = Vorhersage,  $W$  = 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€

# Inhalt

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus**
- 4 Praktische Anwendung
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial

# Einführung

- ▶ „Adaptive Boosting“
- ▶ Entwickelt in den 1990ern von Freund und Schapire, einflussreiches Verfahren für **binäre Klassifikation**
- ▶ Berechnung eines **Lernkoeffizienten** zur Gewichtung eines Lernalgorithmus im Ensemble
- ▶ Nutzung des Lernkoeffizienten zur **exponentiellen Neugewichtung** falsch klassifizierter Datenpunkte

# Einführung

- ▶ „Adaptive Boosting“
- ▶ Entwickelt in den 1990ern von Freund und Schapire, einflussreiches Verfahren für **binäre Klassifikation**
- ▶ Berechnung eines **Lernkoeffizienten** zur Gewichtung eines Lernalgorithmus im Ensemble
- ▶ Nutzung des Lernkoeffizienten zur **exponentiellen Neugewichtung** falsch klassifizierter Datenpunkte



# Einführung

- ▶ „Adaptive Boosting“
- ▶ Entwickelt in den 1990ern von Freund und Schapire, einflussreiches Verfahren für **binäre Klassifikation**
- ▶ Berechnung eines **Lernkoeffizienten** zur Gewichtung eines Lernalgorithmus im Ensemble
- ▶ Nutzung des Lernkoeffizienten zur **exponentiellen Neugewichtung** falsch klassifizierter Datenpunkte

# Einführung

- ▶ „Adaptive Boosting“
- ▶ Entwickelt in den 1990ern von Freund und Schapire, einflussreiches Verfahren für **binäre Klassifikation**
- ▶ Berechnung eines **Lernkoeffizienten** zur Gewichtung eines Lernalgorithmus im Ensemble
- ▶ Nutzung des Lernkoeffizienten zur **exponentiellen Neugewichtung** falsch klassifizierter Datenpunkte

# Vereinfachte Sicht auf den Algorithmus

---

---

**Input:** Datensatz, Lernalgorithmus

Initialisiere Gewichte des Datensatzes

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

    Trainiere schwache Lerner mit gewichtetem Datensatz

    Bestimme Fehler der Lerner

    Wähle schwachen Lerner mit geringstem Fehler

    Berechne Lernkoeffizienten

    Gewichte Datenpunkte neu

**Output:** Starker Lerner (Ensemble)

---



# Notation

- ▶  $\mathcal{X}$  : Menge der **Features**
- ▶  $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- ▶  $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}$ ,  $i = 1, \dots, m$
- ▶ Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- ▶  $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}$ ,  $h_j : X \rightarrow \mathcal{Y}$ ,  $h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- ▶ Anzahl der **Trainingsiterationen**  $T$
- ▶ bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert



# Notation

- ▶  $\mathcal{X}$  : Menge der **Features**
- ▶  $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- ▶  $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}, i = 1, \dots, m$
- ▶ Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- ▶  $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}, h_j : \mathcal{X} \rightarrow \mathcal{Y}, h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- ▶ Anzahl der **Trainingsiterationen**  $T$
- ▶ bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert

# Notation

- $\mathcal{X}$  : Menge der **Features**
- $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}, i = 1, \dots, m$
- Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}, h_j : \mathcal{X} \rightarrow \mathcal{Y}, h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- Anzahl der **Trainingsiterationen**  $T$
- bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert



# Notation

- ▶  $\mathcal{X}$  : Menge der **Features**
- ▶  $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- ▶  $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}$ ,  $i = 1, \dots, m$
- ▶ Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- ▶  $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}$ ,  $h_j : \mathcal{X} \rightarrow \mathcal{Y}$ ,  $h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- ▶ Anzahl der **Trainingsiterationen**  $T$
- ▶ bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert

# Notation

- ▶  $\mathcal{X}$  : Menge der **Features**
- ▶  $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- ▶  $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}$ ,  $i = 1, \dots, m$
- ▶ Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- ▶  $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}$ ,  $h_j : \mathcal{X} \rightarrow \mathcal{Y}$ ,  $h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- ▶ Anzahl der **Trainingsiterationen**  $T$
- ▶ bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert



# Notation

- ▶  $\mathcal{X}$  : Menge der **Features**
- ▶  $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- ▶  $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}, i = 1, \dots, m$
- ▶ Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- ▶  $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}, h_j : \mathcal{X} \rightarrow \mathcal{Y}, h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- ▶ Anzahl der **Trainingsiterationen**  $T$
- ▶ bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert



# Notation

- ▶  $\mathcal{X}$  : Menge der **Features**
- ▶  $\mathcal{Y}$  : Menge der **Labels** ( $\mathcal{Y} = \{-1, +1\}$  bei binärer Klassifikation)
- ▶  $D$  : **Trainingdatensatz** der Form  $D = \{(\mathbf{x}_i, y_i)\}$ ,  $i = 1, \dots, m$
- ▶ Modell wird auf  $D$  durch **Lernalgorithmus**  $\mathcal{L}$  (meistens *Decision Stump*) trainiert und gibt
- ▶  $n$  **Hypothesen**  $(h_j)_{j \in \mathcal{I}}$ ,  $h_j : X \rightarrow \mathcal{Y}$ ,  $h_j(\mathbf{x}) = y$  mit  $\mathcal{I} = \{1, \dots, n\}$  zurück
- ▶ Anzahl der **Trainingsiterationen**  $T$
- ▶ bei jeder Iteration wird  $D$  um **Gewichte**

$$w_i^{(t)}$$

mit  $i = 1, \dots, m$  und  $t = 1, \dots, T$  erweitert

# Initialisierung der Gewichte

- ▶ Zu Beginn sind die Gewichte gleich verteilt

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

- ▶ Die Summe der Gewichte ist stets 1

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

# Initialisierung der Gewichte

- ▶ Zu Beginn sind die Gewichte gleich verteilt

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

- ▶ Die Summe der Gewichte ist stets 1

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

# Training der schwachen Lerner

- **Trainiere** pro Iteration  $n$  **schwache Lerner** (für jedes Feature zwei, je mit umgekehrter Polarität)

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

mit  $w^{(t)}$  als Gewichte der  $t$ -ten Iteration

- Ziel: gewichteten **Fehler minimieren**

$$\varepsilon_j = \sum_{i=1}^m w_i^{(t)} \cdot I(y_i \neq h_j(\mathbf{x}_i)), \quad j = 1, \dots, n$$

Wähle Lerner  $h_j$  aus der Folge mit **geringstem Fehler**  $\varepsilon_j$  als  $h_t$

- Dabei bezeichnet  $I$  die **Indikatorfunktion**

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

# Training der schwachen Lerner

- **Trainiere** pro Iteration  $n$  **schwache Lerner** (für jedes Feature zwei, je mit umgekehrter Polarität)

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

mit  $w^{(t)}$  als Gewichte der  $t$ -ten Iteration

- Ziel: gewichteten **Fehler minimieren**

$$\varepsilon_j = \sum_{i=1}^m w_i^{(t)} \cdot I(y_i \neq h_j(\mathbf{x}_i)), \quad j = 1, \dots, n$$

Wähle Lerner  $h_j$  aus der Folge mit **geringstem Fehler**  $\varepsilon_j$  als  $h_t$

- Dabei bezeichnet  $I$  die **Indikatorfunktion**

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



# Training der schwachen Lerner

- **Trainiere** pro Iteration  $n$  **schwache Lerner** (für jedes Feature zwei, je mit umgekehrter Polarität)

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

mit  $w^{(t)}$  als Gewichte der  $t$ -ten Iteration

- Ziel: gewichteten **Fehler minimieren**

$$\varepsilon_j = \sum_{i=1}^m w_i^{(t)} \cdot I(y_i \neq h_j(\mathbf{x}_i)), \quad j = 1, \dots, n$$

Wähle Lerner  $h_j$  aus der Folge mit **geringstem Fehler**  $\varepsilon_j$  als  $h_t$

- Dabei bezeichnet  $I$  die **Indikatorfunktion**

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

# Berechnung des Lernkoeffizienten

## ► Exponentieller Verlust

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

## ► Herleitung: führe Lernkoeffizienten $\alpha_t$ ein

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



# Berechnung des Lernkoeffizienten

## ► Exponentieller Verlust

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

## ► Herleitung: führe Lernkoeffizienten $\alpha_t$ ein

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



# Berechnung des Lernkoeffizienten

$$y_i = h_t(x_i) \implies y_i h_t(x_i) = 1$$

$\rightsquigarrow$  Beitrag zum Verlust:  $w_i^{(t)} e^{-\alpha_t}$

$$y_i \neq h_t(x_i) \implies y_i h_t(x_i) = -1$$

$\rightsquigarrow$  Beitrag zum Verlust:  $w_i^{(t)} e^{\alpha_t}$

► Minimieren von  $L(h_t)$ :

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

$$\frac{dL(h_t)}{d\alpha_t} = -e^{-\alpha_t} \sum_{y_i=h_t(x_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(x_i)} w_i^{(t)} = 0$$



# Berechnung des Lernkoeffizienten

$$y_i = h_t(x_i) \implies y_i h_t(x_i) = 1$$

$\rightsquigarrow$  Beitrag zum Verlust:  $w_i^{(t)} e^{-\alpha_t}$

$$y_i \neq h_t(x_i) \implies y_i h_t(x_i) = -1$$

$\rightsquigarrow$  Beitrag zum Verlust:  $w_i^{(t)} e^{\alpha_t}$

► Minimieren von  $L(h_t)$ :

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

$$\frac{dL(h_t)}{d\alpha_t} = -e^{-\alpha_t} \sum_{y_i=h_t(x_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(x_i)} w_i^{(t)} = 0$$



# Berechnung des Lernkoeffizienten

$$y_i = h_t(x_i) \implies y_i h_t(x_i) = 1$$

$\rightsquigarrow$  Beitrag zum Verlust:  $w_i^{(t)} e^{-\alpha_t}$

$$y_i \neq h_t(x_i) \implies y_i h_t(x_i) = -1$$

$\rightsquigarrow$  Beitrag zum Verlust:  $w_i^{(t)} e^{\alpha_t}$

► Minimieren von  $L(h_t)$ :

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

$$\frac{dL(h_t)}{d\alpha_t} = -e^{-\alpha_t} \sum_{y_i=h_t(x_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(x_i)} w_i^{(t)} = 0$$

# Berechnung des Lernkoeffizienten

$$\Leftrightarrow e^{2\alpha_t} = \frac{\sum_{y_i=h_t(x_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(x_i)} w_i^{(t)}}$$

$$\Leftrightarrow \alpha_t = \frac{1}{2} \ln \left( \frac{\sum_{y_i=h_t(x_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(x_i)} w_i^{(t)}} \right)$$

► Da  $\sum_{y_i \neq h_t(x_i)} w_i^{(t)} = \varepsilon_t$  und  $\sum_{y_i=h_t(x_i)} w_i^{(t)} = 1 - \varepsilon_t$

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



# Berechnung des Lernkoeffizienten

$$\Leftrightarrow e^{2\alpha_t} = \frac{\sum_{y_i=h_t(x_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(x_i)} w_i^{(t)}}$$

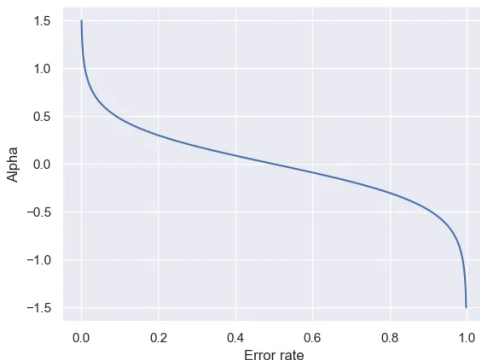
$$\Leftrightarrow \alpha_t = \frac{1}{2} \ln \left( \frac{\sum_{y_i=h_t(x_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(x_i)} w_i^{(t)}} \right)$$

► Da  $\sum_{y_i \neq h_t(x_i)} w_i^{(t)} = \varepsilon_t$  und  $\sum_{y_i=h_t(x_i)} w_i^{(t)} = 1 - \varepsilon_t$

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

# Berechnung des Lernkoeffizienten

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





# Aktualisierung der Gewichte

- Neue Gewichte der Daten für den nächsten Durchlauf berechnen

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

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

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

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





# Aktualisierung der Gewichte

- Neue Gewichte der Daten für den nächsten Durchlauf berechnen

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

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

$$\begin{aligned}
 Z_t &= \sum_{j=1}^m w_j^{(t+1)} && \text{(Normalisierungsfaktor)} \\
 w_i^{(t+1)} &= \frac{w_i^{(t+1)}}{Z_t}
 \end{aligned}$$

# Das Ergebnis des Algorithmus

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

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

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



# Der vollständige Algorithmus I

---

**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  $\varepsilon_j$  als  $h_t$ 
```

```
    // Berechne den Lernerkoeffizienten
```

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

```
    // 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
     $w_i^{(t+1)} \leftarrow w_i^{(t)} \cdot e^{-\alpha_t}$ 
else
     $w_i^{(t+1)} \leftarrow w_i^{(t)} \cdot e^{\alpha_t}$ 

// Normalisiere Gewichte
 $Z_t \leftarrow \sum_{j=1}^m w_j^{(t+1)}$ 
for  $i = 1$  to  $m$  do
     $w_i^{(t+1)} \leftarrow \frac{w_i^{(t+1)}}{Z_t}$ 

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

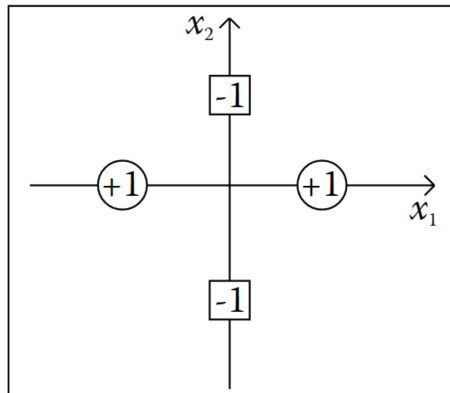
// Ende des Algorithmus
```

---



# Beispiel

## Das XOR-Problem (eine Variation)



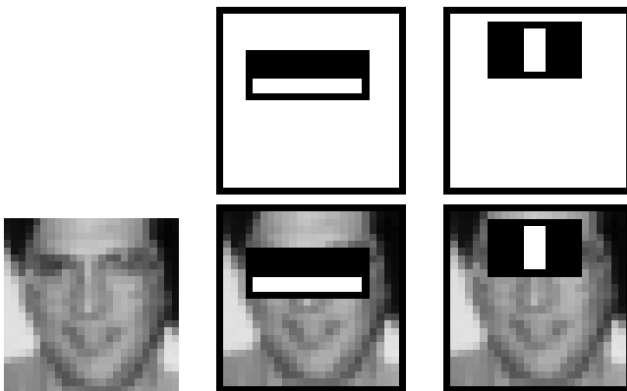
# Inhalt

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung**
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial



# Praktische Anwendung und Beispiele

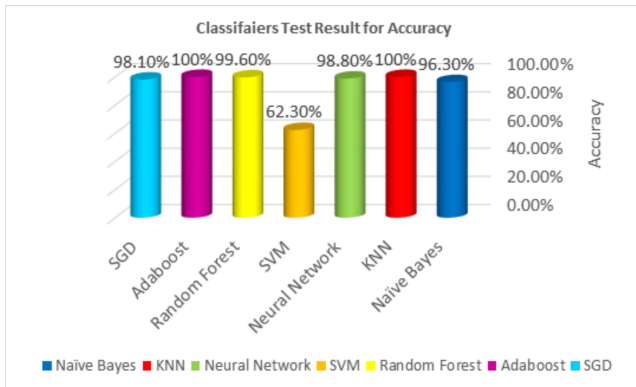
## Bilderkennung und Computervision: Gesichtserkennung





# Praktische Anwendung und Beispiele

## Textklassifikation und Natural Language Processing: Erkennung von Spam-Mail

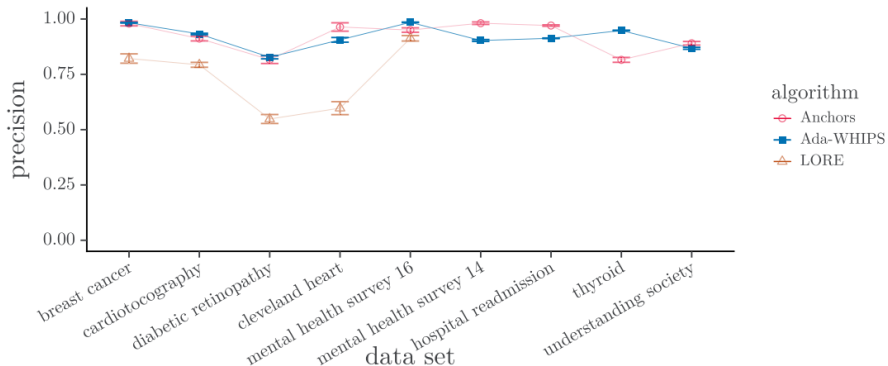






# Praktische Anwendung und Beispiele

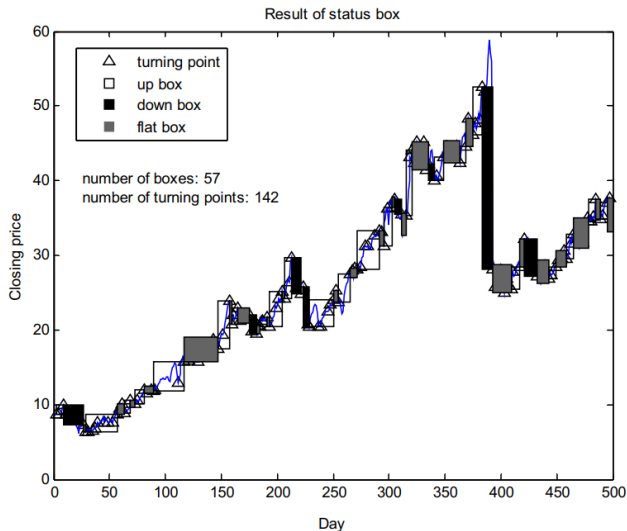
**Medizinische Diagnostik:** Risiko/Erkennung von Krankheiten basierend auf Patientendaten





# Praktische Anwendung und Beispiele

## Finanzwesen: Vorhersage von Aktienkursbewegungen



# Inhalt

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung
- 5 Vor- und Nachteile**
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial



# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für **verrauschte Daten** und **Ausreißer**
- Training auf großen Datensätzen kann **zeitintensiv** sein
- Hauptsächlich für **binäre Klassifikation** ausgelegt

# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für verrauschte Daten und Ausreißer
- Training auf großen Datensätzen kann zeitintensiv sein
- Hauptsächlich für binäre Klassifikation ausgelegt

# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für verrauschte Daten und Ausreißer
- Training auf großen Datensätzen kann zeitintensiv sein
- Hauptsächlich für binäre Klassifikation ausgelegt



# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für verrauschte Daten und Ausreißer
- Training auf großen Datensätzen kann zeitintensiv sein
- Hauptsächlich für binäre Klassifikation ausgelegt

# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für verrauschte Daten und Ausreißer
- Training auf großen Datensätzen kann zeitintensiv sein
- Hauptsächlich für binäre Klassifikation ausgelegt



# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für **verrauschte Daten und Ausreißer**
- Training auf großen Datensätzen kann **zeitintensiv** sein
- Hauptsächlich für **binäre Klassifikation** ausgelegt

# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für **verrauschte Daten und Ausreißer**
- Training auf großen Datensätzen kann **zeitintensiv** sein
- Hauptsächlich für **binäre Klassifikation** ausgelegt

# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für **verrauschte Daten und Ausreißer**
- Training auf großen Datensätzen kann **zeitintensiv** sein
- Hauptsächlich für **binäre Klassifikation** ausgelegt

# Vor- und Nachteile von AdaBoost

## Vorteile:

- + Benutzerfreundlich
- + Flexibel
- + Identifiziert automatisch wichtige Features
- + Neigt weniger zum Overfitting

## Nachteile:

- Anfällig für **verrauschte Daten und Ausreißer**
- Training auf großen Datensätzen kann **zeitintensiv** sein
- Hauptsächlich für **binäre Klassifikation** ausgelegt

# Inhalt

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen**
- 7 Literatur und Zusatzmaterial

# Erweiterungen und Variationen von AdaBoost

- ▶ Ursprünglich für binäre Klassifikation entwickelt, durch verschiedene Erweiterungen für diverse Problemstellungen adaptiert
- ▶ „AdaBoost.M1“ und „SAMME“ für **Multiklassen-Probleme**
- ▶ **Kosten-sensitives** AdaBoost
- ▶ Neben Decision Stumps kann AdaBoost mit **SVMs, Neuronalen Netzen und anderen Classifiern** kombiniert werden

# Erweiterungen und Variationen von AdaBoost

- ▶ Ursprünglich für binäre Klassifikation entwickelt, durch verschiedene Erweiterungen für diverse Problemstellungen adaptiert
- ▶ „AdaBoost.M1“ und „SAMME“ für **Multiklassen-Probleme**
- ▶ **Kosten-sensitives** AdaBoost
- ▶ Neben Decision Stumps kann AdaBoost mit **SVMs, Neuronalen Netzen und anderen Classifiern** kombiniert werden

# Erweiterungen und Variationen von AdaBoost

- ▶ Ursprünglich für binäre Klassifikation entwickelt, durch verschiedene Erweiterungen für diverse Problemstellungen adaptiert
- ▶ „AdaBoost.M1“ und „SAMME“ für **Multiklassen-Probleme**
- ▶ **Kosten-sensitives** AdaBoost
- ▶ Neben Decision Stumps kann AdaBoost mit **SVMs, Neuronalen Netzen und anderen Classifiern** kombiniert werden



# Erweiterungen und Variationen von AdaBoost

- ▶ Ursprünglich für binäre Klassifikation entwickelt, durch verschiedene Erweiterungen für diverse Problemstellungen adaptiert
- ▶ „AdaBoost.M1“ und „SAMME“ für **Multiklassen-Probleme**
- ▶ **Kosten-sensitives** AdaBoost
- ▶ Neben Decision Stumps kann AdaBoost mit **SVMs, Neuronalen Netzen und anderen Classifiern** kombiniert werden



# Erweiterungen und Variationen von AdaBoost

- ▶ **Robuste** Varianten minimieren die Auswirkung von Ausreißern.
- ▶ **Online** AdaBoost aktualisiert Modelle ohne Neutrainierung.
- ▶ **Direkte Feature Auswahl:** Wählt während des Trainings aus, welche Features wichtig sind und betrachtet nur diese ~> **schnelleres Training**
- ▶ Variationen, welche die **Interpretierbarkeit und Erklärbarkeit** verbessern



# Erweiterungen und Variationen von AdaBoost

- ▶ **Robuste** Varianten minimieren die Auswirkung von Ausreißern.
- ▶ **Online** AdaBoost aktualisiert Modelle ohne Neutrainierung.
- ▶ **Direkte Feature Auswahl:** Wählt während des Trainings aus, welche Features wichtig sind und betrachtet nur diese ~> **schnelleres Training**
- ▶ Variationen, welche die **Interpretierbarkeit und Erklärbarkeit** verbessern

# Erweiterungen und Variationen von AdaBoost

- ▶ **Robuste** Varianten minimieren die Auswirkung von Ausreißern.
- ▶ **Online** AdaBoost aktualisiert Modelle ohne Neutrainierung.
- ▶ **Direkte Feature Auswahl:** Wählt während des Trainings aus, welche Features wichtig sind und betrachtet nur diese  $\rightsquigarrow$  **schnelleres Training**
- ▶ Variationen, welche die **Interpretierbarkeit und Erklärbarkeit** verbessern

# Erweiterungen und Variationen von AdaBoost

- ▶ **Robuste** Varianten minimieren die Auswirkung von Ausreißern.
- ▶ **Online** AdaBoost aktualisiert Modelle ohne Neutrainierung.
- ▶ **Direkte Feature Auswahl:** Wählt während des Trainings aus, welche Features wichtig sind und betrachtet nur diese  $\rightsquigarrow$  **schnelleres Training**
- ▶ Variationen, welche die **Interpretierbarkeit und Erklärbarkeit** verbessern

# Inhalt

- 1 Einleitung
- 2 Grundlagen des Boosting
- 3 Der AdaBoost Algorithmus
- 4 Praktische Anwendung
- 5 Vor- und Nachteile
- 6 Erweiterungen und Variationen
- 7 Literatur und Zusatzmaterial**

# 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.



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.



Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou.

Multi-class adaboost.

*Statistics and its Interface*, 2(3):349–360, 2009.



# Literatur II



Hamed Masnadi-Shirazi and Nuno Vasconcelos.

Cost-sensitive boosting.

*IEEE Transactions on pattern analysis and machine intelligence*,  
33(2):294–309, 2010.



Manish Panwar, Jayesh Rajesh Jogi, Mahesh Vijay Mankar,

Mohamed Alhassan, and Shreyas Kulkarni.

Detection of Spam Email.

*AJISE*, 1, 2022.



Paul Viola and Michael Jones.

Fast and robust classification using asymmetric adaboost and a  
detector cascade.

*Advances in neural information processing systems*, 14, 2001.





# Literatur III



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.



Jianxin Wu, James M Rehg, and Matthew Mullin.

Learning a rare event detection cascade by direct feature selection.

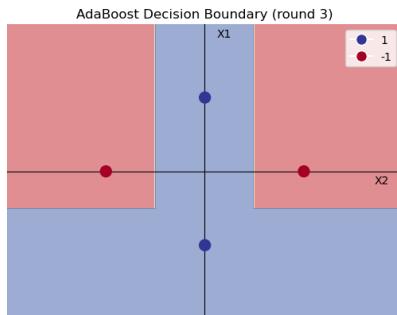
*Advances in Neural Information Processing Systems*, 16, 2003.

# Literatur IV



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  $\text{\LaTeX}$  auf [GitHub](#).

Danke

Vielen Dank für die Aufmerksamkeit!