# KE und Lernen in Spielen

## Classification and Regression Trees

Markus Müller

# Gliederung

- Lernen
- Entscheidungsbäume
- Induktives Lernen von Bäumen
  - ID3 Algorithmus
  - Einfluß der Beispielmenge auf den Baum
  - Möglichkeiten zur Verbesserung
- Anwendung in Spielen
- Demonstration

# Lernen allgemein

- Was bedeutet Lernen?
  - Das Abspeichern von Faktenwissen ist nicht gemeint!
  - Einschätzen (Klassifizieren) einer Situation und Durchführen einer geeigneten Reaktion.
  - Wählen einer anderen Reaktion falls in der Vergangenheit Mißerfolge → dazu lernen!
  - Vorhersagen einer Situation nach dem Durchführen einer Aktion.

# Lernen in Spielen

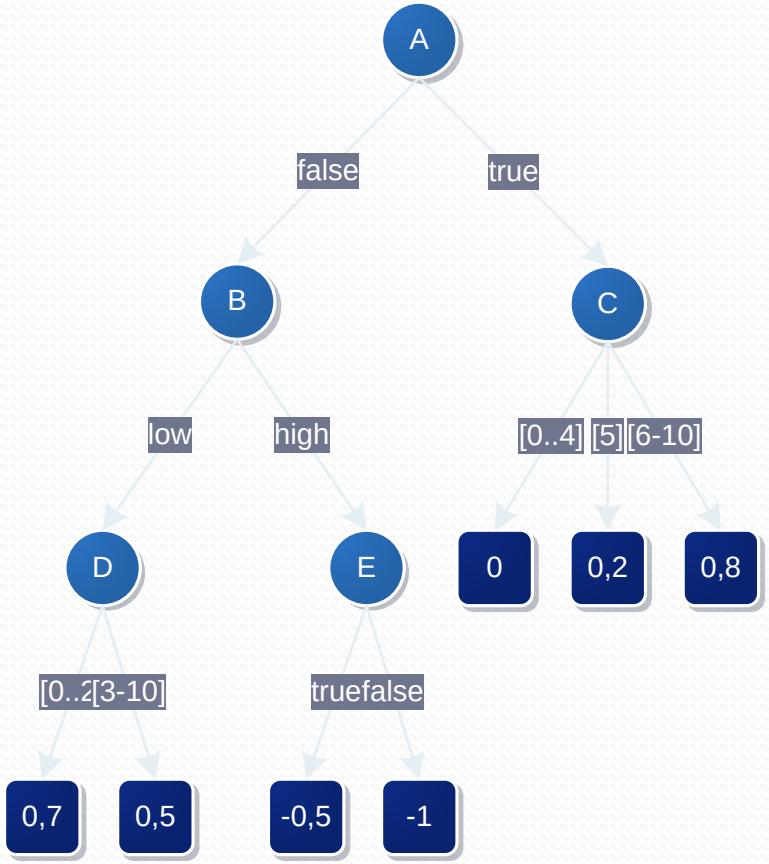
- NPC Steuerung
  - Handlungen in unbekannten Situationen
  - An Situation angepaßte Taktik
    - Herausfordernde Gegner
  - Anpassung an Taktik des Spielers
    - Kooperative NPCs
- Modellbildung
  - Erfahrungen
  - Beispieldaten

# Maschinelles Lernen

- Grundlage: Beispiele
  - Bewertete Datensätze
- Lernverfahren
  - Bilden einer Hypothese auf Grund der Beispiele
- Hypothesen
  - Hier: Entscheidungsbäume
- Gelernt wird also nicht die Klassifizierung direkt!

# Entscheidungsbäume

- Berechnen den Wert eines Ausgabeattributs für eine Menge von Eingabeattributen
- Grundstruktur: Baum
  - Knoten = Entscheidungen
  - Kante = Ergebnisse
  - Blätter = Ausgabewert
- Zwei Ausprägungen
  - Classification Trees
  - Regression Trees



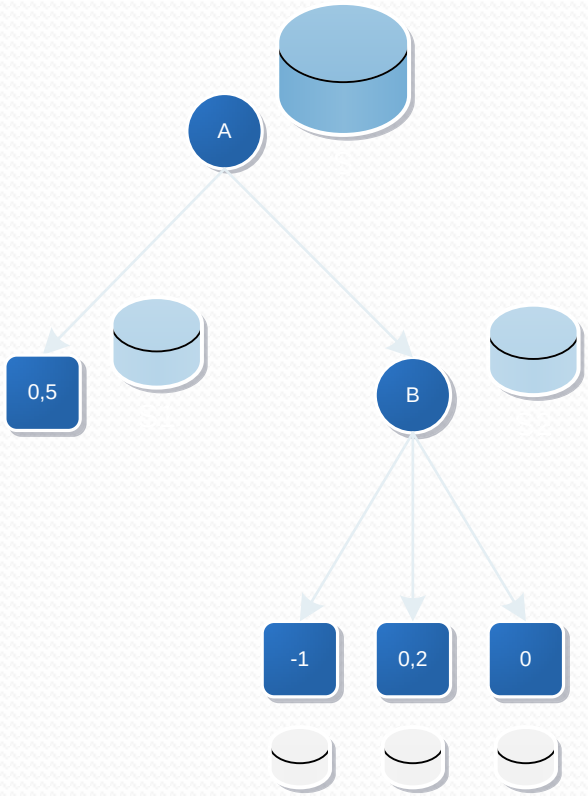
# EB: Datenmenge

- Menge von Eingabeattributen (predictor variables)
  - Symbolische Werte oder Gleitkommawerte
- Ausgabeattribut (response variable)
  - Classification Tree: symbolischer Wert
  - Regression Tree: Gleitkommawert
- Fast jedes Attribut kann das Ausgabeattribut werden, je nach Aufgabenstellung

| Gewicht | Schüsse / Minute | Kapazität (Stk.) | Entfernung (m) | Typ | Schaden |
|---|---|---|---|---|---|
| Leicht | 47 | 10 | 40 | Handfeuerwaffe | 5% |
| Schwer | 200 | 500 | 100 | Maschinengewehr | 10% |
| Sehr leicht | 6 | 6 | 25 | Handfeuerwaffe | 4% |
| Sehr schwer | 280 | 1000 | 200 | Maschinengewehr | 13% |

# Entscheidungsbäume

- Jede Entscheidung teilt die Beispielmenge in disjunkte Untermengen auf.
  - Jede Entscheidung hat für ein spezifisches Beispiel immer nur ein (!) Ergebnis.
  - (C in [0..4]) und (C in [2..5]) auf der gleichen Ebene also nicht möglich!



# Bearbeiten eines Datensatzes

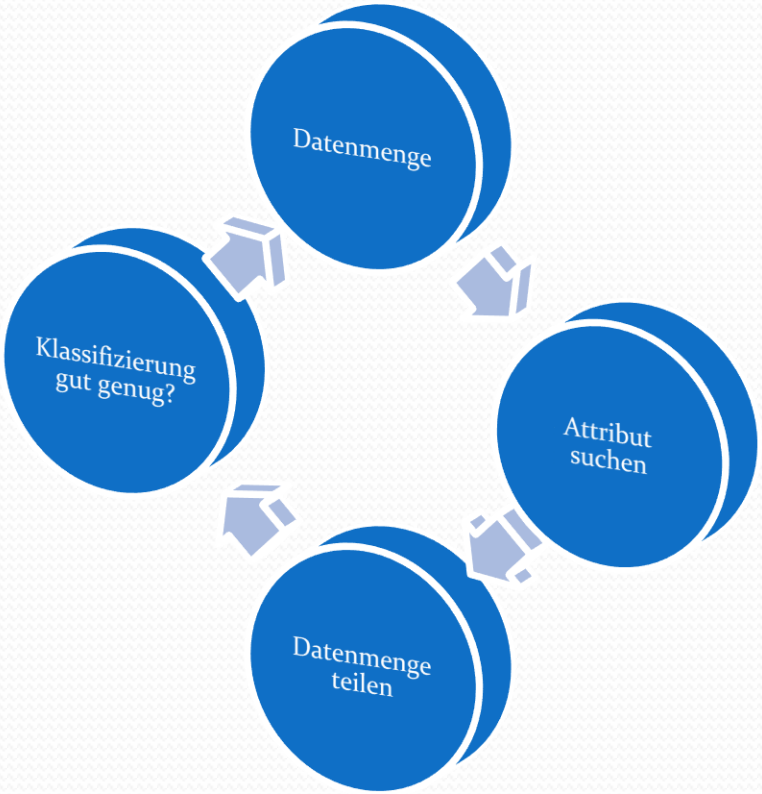
- Bei gegebenem Baum kann der Wert des Ausgabeattributs durch die bekannten Werte der Eingabeattribute hergeleitet werden.
- Herleitung durch traversieren durch den Baum von der Wurzel zum entsprechenden Blatt

**Pseudocode**

```
Node = root
Repeat
  result = node.evaluate(sample)
  for each branch from node
    if branch.match(result)
      node = branch.child
    end if
  end for
until node is a leaf
return leaf class or value
```

# Induktives Lernen, ID3

- Rekursive Partitionierung
  - ID3 Algorithmus von R. Quinlan, 1975
- Die Beispielmenge wird in grob klassifizierte Untermengen geteilt.
  - Anzahl der Untermengen abhängig vom Attribut an dem geteilt wird
- Teilen wiederholen bis die Klassifizierung perfekt bzw. „gut genug“ ist.



# ID3-Algorithmus

- Start mit leerem Baum und voller Beispielmenge
- Attribut finden, dass die Datenmenge am Besten klassifiziert.
- Entscheidungsknoten für dieses Attribut erstellen und die Datenmenge entsprechend aufteilen.
- Beenden
  - Falls Klassifizierung perfekt oder nicht mehr zu verbessern
  - Falls keine Daten mehr

**Pseudocode**

```
Function partition(dataset, node)
  if not create_decision(dataset,
node)
    return
  end if
  for each sample in dataset
    result = node.evaluate(sample)
    subset[result].add(sample)
  end for
  for each result in subset
    partition(subset, child)
    node.add(branch, result)
    branch.add(child)
  end for
End function
```

# ID3-Algorithmus

- Welches Attribut klassifiziert die Datenmenge am Besten?
  - Berechnen der „Unreinheit“ (impurity / entropy) der Mengen in Bezug auf das Ausgabeattribut.
    - 0 falls nur Datensätze mit gleichem Ausgabeattribut
    - 1 falls gleichmäßige Verteilung aller möglichen Werte
  - Berechnen der möglichen Verbesserung (information gain), falls an diesem Attribut geteilt wird.
  - Wählen des Attributes mit der größten Verbesserungsmöglichkeit.

# ID3-Algorithmus

- S: Beispielmenge
- t: Ausgabeattribut
- Value(t): Werte des Ausgabeattributs

Entropy

$$I(S) = -\sum_{i \in Value(t)} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Entropy nach Teilung

$$E(a) = \sum_{v \in Value(a)} \frac{|S_v|}{|S|} I(S_v)$$

Information gain

$$Gain(S,a) = I(S) - E(a)$$

**Pseudocode**

```
Function create_decision(dataset,
node)
  max=0
  entropy = compute_entropy(dataset)
  for each attribute in dataset
    e = entropy -
compute_entropy_split(attribute,
dataset)
    if e > max
      max = e
      best = attribute
    end if
  end for

If best
  node.evaluation =
create_test(attribute)
Else
  node.class = find_class(dataset)
End if
```

# Einfluß der Beispielmenge

- Der gelernte Baum ist sehr spezifisch für die verwendete Beispielmenge
  - Der Baum sollte das Problem und nicht die verwendeten Daten repräsentieren.
  - Kann für andere Beispiele zu spezifisch sein → overfitting
- Lösung: Verwenden einer zweiten Datenmenge
  - 1. Trainingsdaten für ID3-Algorithmus
  - 2. Validierungsdaten um Baum zu generalisieren

# Generalisieren / Pruning

- Idee: Falls die Klassifizierung ohne einen Zweig des Baumes gleich gut oder besser ist als mit ihm, wird der Zweig aus dem Baum entfernt.
- Pruning wird nach dem Lernen des Baums ausgeführt.
- Benötigt eine andere Datenmenge, da Trainingsdaten keine Änderungen auslösen würden.

# Generalisieren / Pruning

- Für jeden Entscheidungsknoten wird ausgerechnet welchen Wert das Ausgabeattribut hätte, wenn er ein Blatt wäre
  - Majorität oder Durchschnitt der enthaltenen Werte
- Bearbeiten aller Datensätze der Validierungs-menge.
- Für jeden Knoten wird festgehalten wie oft er einen Datensatz korrekt klassifiziert hat.
- Falls ein Knoten einen höheren Wert hat als die Summe seiner Nachfolger sind die Nachfolger überflüssig und können gelöscht werden.

# Bagging und Boosting

- Idee: Schwache Klassifizierer werden kombiniert um so bessere zu erhalten.
- Durch Änderung bzw. Teilung der Trainingsdaten können verschiedene Bäume erzeugt werden
- Bagging
  - Die Klassifizierung mit den meisten Treffern wird ausgegeben.
- Boosting
  - Die einzelnen Klassifizierer werden zudem je nach Leistung auf den Validierungsdaten gewichtet.

# Beispiel: Quake, Waffenwahl

- Idee:
  - Entscheidungsbaum für zu wählende Waffe
  - Problem: Waffe wurde u.U. noch gar nicht gefunden
- Besser:
  - Ein Entscheidungsbaum pro Waffe (Regression Tree)
  - Gibt an, wie gut eine Waffe „passt“

# Beispiel: Quake, Waffenwahl

- Eingabeattribute:
  - Distanz zum Gegner [near, medium, far]
  - Eigene Lebensenergie [low, high]
  - Munitionsvorrat [low, medium, high]
  - Bewegungsrichtung [forward, backward]
- Ausgabeattribut:
  - Sollte die Waffe verwendet werden [0..1]
- Auswahl
  - Waffe mit höchstem Wert, die schon gefunden wurde

# Beispiel: Black & White

- Die autonome Kreatur hat Verlangen nach
  - Essen
  - Trinken
  - Schlaf
  - Gesundheit
- Sie hat eine Meinung darüber welche Objekte zur Befriedigung welches Verlangens geeignet sind.
  - Regression Tree

# Demonstration

- Eigene Implementierung des ID3 Algorithmus

# Quellen


- David M. Bourg, Glenn Seemann: *AI for Game Developers*, O'Reilly, 2004
- Alex J. Champandard: *AI Game Development*, New Riders Publishing, 2003
- Knut Hartmann: Echtzeittechniken für Computerspiele – Lernfähige Agenten, Uni Magdeburg, 2005
- Richard Evens: The Use of AI Techniques in Black & White, Lionhead
- Prof. Jantke: Theorie des Algorithmischen Lernens, TU Darmstadt