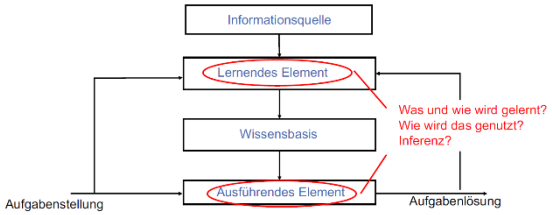
Maschinelles Lernen I

Lernzusammenfassung zur Vorlesung am KIT
Lukas Bach - lbach@outlook.de - lukasbach.com

1 Einführung

- Was ist Lernen?
  - Lernen von Entscheidungen, Aktionsfolgen, Beschreibungen, Modellen…
  - Lernen motorischer/kognitiver Fähigkeiten durch Anweisungen/Training
  - Neuorganisation/Transformation von Wissen
- Maschinelles Lernen
  - Ein System lernt aus Erfahrung E in Hinblick auf eine Klasse von Aufgaben T und einem Performancemaß P, wenn seine Leistungen bei Aufgaben aus T gemessen mit P durch Erfahrung aus E steigt.
    Ein lernendes System generiert eine oder mehrere Lösungshypothese(n) H um Aufgaben T zu lösen.
  - Beispiel: Schach spielen lernen.
    - T = Schach spielen
    - P = % der gewonnenen Spiele
    - E = Spiele gegen sich selbst
    - H = Modell um Spielzüge anhand der aktuellen Situation zu erzeugen
- Komponenten eines lernenden Systems



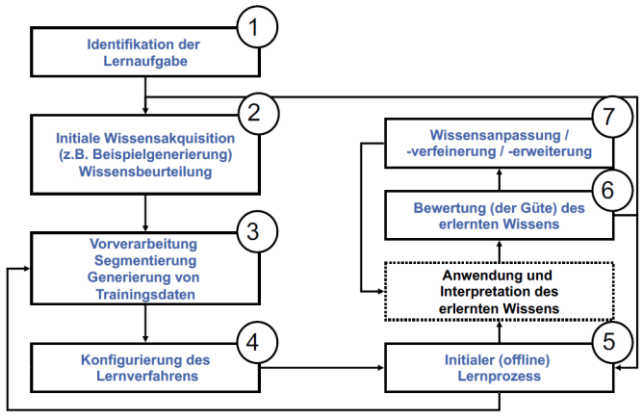
  -
- Deduktiver Schluss
  - Aus einer Menge von Formeln A folgt B
    ⇔ Es gibt eine Folge von Regeln, um B herzuleiten.
  - Beispiele: Modus Ponens, Instantiierung
- Abduktion
  - H folgt aus Hintergrundwissen B und Beobachtungen D abduktiv

$$\Leftrightarrow B \cup H \mapsto D$$

- Induktiver Schluss
  - Geg. Menge D von Grundbeispielen. Hypothese H folgt induktiv aus D und Grundwissen B

$$\Leftrightarrow B \cup H \mapsto D, B \not\mapsto D, B \cup D \not\mapsto \neg H$$

- Wissensrepräsentation
  - Assoziierte Paare (In/Output)
  - Parameter in algebraischen Ausdrücken (Funktionale Form von numerischen Parametern, zB Gewichtsmatrix)
  - Entscheidungsbäume (Klassendiskriminierung)
  - Formale Grammatiken
  - Produktionsregeln
  - Formale logikbasierte Ausdrücke
  - Graphen/Netzwerke
  - Probabilistische graphische Modelle (Verbundwahrsch. von Zufallsvariablen)
  - Frames, Schemata, Schemantische Netze
  - Prozedurale Kodierung
  - Taxonomien (Hierarchische Klassifikation, Lernen durch Beobachtung)
  - Markov-Ketten (zustandsbasiertes Hintergrundwissen)
  - Kombination von Repräsentationsarten
- Lernen als Prozess



(Siehe Folien F01.57)

- Einordunungskriterien von Lernverfahren

Typ der Inferenz	induktiv vs deduktiv
Ebenen des Lernens	symbolisch vs subsymbolisch
Lernvorgang	überwacht vs unüberwacht
Beispielgebung	inkrementell vs nicht inkr.
Umfang der Bsps	umfachreich vs gering
Hintergrundwissen	empirisch vs axiomatisch

1.1 Vorlesung Übersicht

- Einführung
- Induktives Lernen
- Lerntheorie
- Support Vektor Machine
- Neuronale Netze
- Convolutional Neural Networks (CNN)
- Entscheidungsbäume
- Reinforcement Learning
- Hidden Markov Modelle
- Lernen nach Bayes
- Instanzbasiertes Lernen
- Unüberwachtes Lernen
- Deduktives Lernen
- Evolutionäre Algorithen

2 Induktives Lernen

2.1 Induktion und Deduktion

- *Induktion*: Prozess des plausiblen Schließens vom Speziellen zum Allgemeinen. Basis: Viele zutreffende Fälle.
- *Deduktion*: Prozess des korrekten Schließens vom Allgemeinen zum Speziellen. Basis: Hintergrundwissen (Regeln). Bsp: Modus Ponens.
- Induktion vs Deduktion

Induktion	Deduktion
Wahrheitserweiternd	Wahrheiserhaltend
Grundsatz: induktive Lernhyptothese	Logischer Schluss
Plausibilität	Korrektheit

- Induktive Lernverfahren: Konzeptlernen, Finde *Hypothesen* $h \in H$ aus *Hypothesenraum* H die ein *Zielkonzept c(..)* mit Instanzen des *Instanzraums X* erfüllen anhand *Trainingsmenge* $D \subset X$ als Untermenge des Instanzraums.
- Allgemeine Bezeichnungen: Instanzraum = Merkmalsraum, Zielkonzept = Sollausgabe, Erfüllen des Zielkonzepts = Definition eines minimierten Fehlermaß
- *Induktive Lernhypothese*
  - Jede Hypothese, die die Zielfunktion über einer ausreichend großen Menge von Trainingsbeispielen gut genug approximiert, wird die Zielfunktion auch über unbekannten Beispielen gut approximieren.

2.2 Konzeptlernen als Suche im Hyptothesenraum

- *Konzept*: Beschreibt Untermenge von Objekten aus größerer Menge mittels boolscher Funktion.
  - Bsp: vogel: Tier -> {w,f}. vogel(Storch)=w.
- Konzeptlernen: Schließen auf boolsche Funktion aus Trainingsbeispielen von Input/Output. Instanzen werden meist durch Attribute beschrieben.
- *Konsistenz*: Keine False Positives.
- *Vollständig*: Positive Beispiele werden alle positiv klassifiziert.
- Lerne als Suche im Hypothesenraum.
  - Suche vom Allgemeinen zum Speziellen
    - Gehe von allgemeinster Hypothese aus. Spezialisiere auf negativen Beispielen und ignoriere positive Beispiele.
  - Suche vom Speziellen zum Allgemeinen
    - Gehe von speziellster Hypothese aus. Verallgemeinere auf positiven Beispielen und ignoriere negative Beispiele.
  - Paralleles Anwenden beider Methoden: Version Space

2.3 Specific-to-general Suche

2.3.1 Algorithmus

```
Initialisiere h mit spezifischsten Hypothese in H
Für jedes positive Trainingsbeispiel x
  Für jede Attributseinschränkung ai in h
    Wenn ai von x erfüllt wird:
      Tue nichts.
    Sonst:
      Ersetze ai durch die nächstallgemeinere
      Einschränkung, die durch x erfüllt wird.
Gib die Hypothese aus.
```

2.3.2 Beispiel

Siehe Folien F02.20

2.3.3 Beurteilung

- Wichtiges Prinzip im Konzeptlernen.
- Endhypothese ist auch mit negativen Trainingsbeispielen konsistent, solange Trainingsbeispiele konsistent sind und Zielhypothese in H ist.

2.4 Versionsraum/Candidate-Elimination-Algorithmus

Versionsraum: Der *Versionsraum* $VS_{H,D}$ bzgl. H, D ist die Untermenge der Hypothesen von H, die mit Trainingsbeispielen konsistent und vollständig sind.

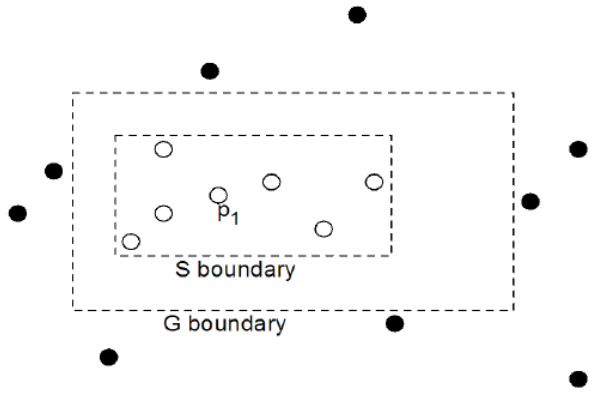
- $VS_{H,D} = \{h \in H : consistent(h, D), compl(h, D)\}$
- Hypothesenraum H, Trainingsmenge D

2.4.1 Algorithmus

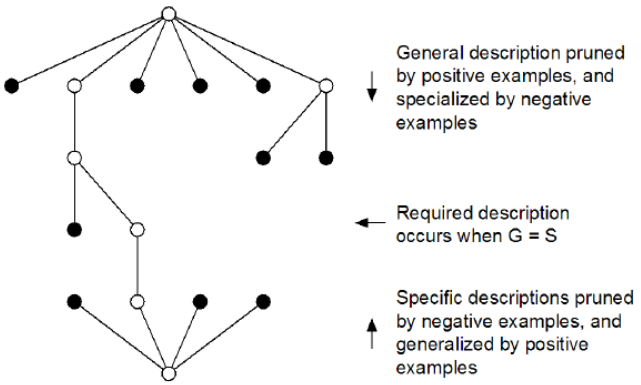
- Halte Menge von spezifischsten und Menge von allgemeinsten Hypothesen.
- Ist n ein negatives Beispiel:
  - Lösche aus spezifischsten Hypothesen die Hypothesen, die n abdecken.
  - Spezialisiere die allgemeinsten Hypothesen soweit, dass sie n nicht abdecken und allgemeiner als eine spezielle Hypothese bleiben.
  - Lösche alle allgemeinsten Hypothesen, die spezifischer als eine andere allgemeine Hypothese sind.
- Ist p ein positives Beispiel:
  - Lösche alle allgemeinsten mit p inkonsistenten Hypothesen.
  - Verallgemeinere alle spezifischen Hypothesen soweit, dass sie p abdecken und spezifischer als eine allgemeine Hypothese bleiben.
  - Lösche alle spezifischen Hypothesen, die allgemeiner als eine andere spezifische Hypothese sind.

2.4.2 Veranschaulichung

Verhältnis spez. zu allg. Hypothese:



Suche im Hypothesenraum:



(●: pruned descriptions)

Beispiel siehe Folien F02.35

2.4.3 Beurteilung

VS konvergiert zur korrekten Hypothese, wenn Beispiele konsistent sind und korrekte Hypothese in H enthalten ist.

Probleme: Rauschen in D, Zielhypothese nicht von Hypothesenrepräsentation abgedeckt.

Wenn mehrere Hypothesen unterschiedlich klassifizieren, zB. probabilistische Ausgabe.

2.5 Notwendigkeit von Vorzugskriterien (Bias)

- Bisher: Wie sehr kann man mit Beispielen verallgemeinern? Der Hypothesenraum ist aber wichtig!
  - Problem: Zielkonzept evtl. nicht im Hypothesenraum.
  - Lösung: Hypothesenraum, der alle möglichen Hypothesen enthält?
- Ein induktives Lernsystem, das keine apriori Annahmen über die Identität des Zielkonzepts macht, hat keine rationale Basis, um unbekannte Instanzen zu klassifizieren.
- Induktives Lernen erfordert Vorannahme „Inductive Bias“
- Bsp: Auswendiglerner haben keine Annahme. Specific-to-General: Alle Instanzen sind negativ-Instanzen bis Gegenteil bekannt ist.
- Je strenger die Vorannahmen, desto mehr unbekannte Instanzen werden erkannt.
- Vorzugskriterien (*Bias*)
  - Vorschrift als Grundlage für entwickelte Hypothesen
  - Beispiele: Verständlichkeit, Klassifikationsgenauigkeit, Messaufwand, Aufwand
  - *Hypothesenraumbias*: Beschränke den Raum von Hypothesen (Polynome, 3-NN…)
  - *Präferenzbias*: Ordne Hypothesenraum. Bevorzuge Hypothesen mit wenig Disjunktionen oder kleinen Entscheidungsbäumen.
  - Problem: Keine Funktion h ist mit allen Trainingsbeispielen kosistent zB wegen Rauschen.
  - Lösungsansätze:
    - Hypothesenraumbias anpassen
      - Komplexe Hypothesen. Aber: Overfitting!
    - Präferenzbias anpassen
      - Bestmögliche Hypothese wählen

3 Lerntheorie

3.1 Definition des Lernens

- Grundsatz von Lernmaschinen
  - Entities should not be multiplied beyxond necessity
  - Löse nie ein Problem komplizierter als nötig, denn die einfachste richtige Erklärung ist die beste.
- Eine *lernende Maschine* wird bestimmt durch Hypothesenraum $H = \{h_\alpha : \alpha \in A\}$ und Lernverfahren (Methode um α_{opt} mit Lernbeispielen zu finden. Dazu Fehlerfunktion und Optimierungsmethode notwendig.)
- Probleme: Welche Maschine? Welcher Hypothesenraum? Welches Lernverfahren? Welche Hypothesen wählen?
- Probleme beim Lernen:
  - *Statistisches Problem* (zu großer Hypothesenraum. Mehrere Hypothesen sind gleich gut)
    - Beispiel: Regression, kann falsch erkannt werden.
  - *Komplexitätsproblem*: Wegen Problemkomplexität kann die optimale Lösung nicht garantiert im Hypothesenraum gefunden werden ⇒ Suboptimale Lsg.
    - Beispiel: Klassen nicht linear trennbar
  - *Repräsentationsproblem*: Hypothesenraum enthält keine ausreichend gute Approximation der Zielfunktion.
    - Beispiel: Keine strikte Klassentrennung möglich.
- Lernproblem
  - $\{Attr_1, \dots, Attr_n\} \times \{true, false\} = Konzept$
  - $R^N \times \{Klasse_1, \dots, Klasse_n\} = Klassifikation$
  - $R^N \times R = Regression/Prognose$

3.2 Fehler

3.2.1 Lernen durch Fehlerminimierung

Fehler: Schätze h so, dass der Fehler $E(h)$ minimal ist mit:

$$E(h_\alpha) = Func((h_\alpha(\vec{x}) \leftrightarrow y), P(\vec{x}, y))$$

- h_α ist die zu lernende Abbildung
- $(\overrightarrow{x_n}, y_n) \in X \times Y$ sind die Lernbeispiele
- $P(\vec{x}, y)$ ist die Wsk der Lernbeispiele
- $\leftrightarrow$ ist der Vergleich zwischen Hypothesenausgabe und Sollausgabe.

Lernen: Wie kann E effektiv minimiert werden?

3.2.2 Empirischer und realer Fehler

- *Realer Fehler* $E(h) = \int l(h(\vec{x}), y) d \cdot P(\vec{x}, y)$ bzgl aller realer Daten ist nicht berechenbar, aber abschätzbar.
- *Empirischer Fehler* $E_D(h) = \frac{1}{|D|} \sum_{(\vec{x},y) \in D} l(h(\vec{x}), y)$
  - Lernfehler für D = Lerndaten
  - Verifikationsfehler für D = Verifikationsdaten
  - Generalisierungsfehler für D = Testdaten
  - Erfordert unabhängige identische Verteilung über Datensätze
- Empirische Fehlerminimierung kann zu Overfitting führen

3.3 Overfitting

Klar. Siehe Folien F03.15

3.4 Hypothesenbewertung, Modellauswahl

3.4.1 Modellauswahl

- Komplexe Maschine => schlechtes Modell: Zu viele Parameter, aber wenig komplexe Maschine => schlechtes Modell: Wenig Parameter
- Metriken beschreiben Modellgüte. Methoden je nach Aufgabenstellung
- Ergebnisklassen der Klassifikation: True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN)
- Konfusionsmatrix: Wahrheitsmatrix für 4 Ergebnisklassen, siehe F03.19
- *Metriken* (Pfeile geben an ob Wert groß oder klein sein sollte)
  - ↓Klassifikationsfehler: $\frac{errors}{total} = \frac{FP+FN}{TP+FN+FP+TN}$
  - ↑Güte (=1-Fehler): $\frac{correct}{total} = \frac{TP+TN}{TP+FN+FP+TN}$
  - ↓FP-Rate/False Alarm Rate FA: $FPR = \frac{FP}{FP+TN}$
  - ↓FN-Rate/Miss Rate MR: $FNR = \frac{FN}{TP+FN} = 1 - TPR$
  - ↑Genauigkeit/Precision: $P = \frac{TP}{TP+FP}$
  - ↑TP-Rate/Positive Rückmeldung/Recall: $TPR = R = \frac{TP}{TP+FN} = 1 - FNR$
  - ↑F1-Maß: $F_1 = \frac{2}{\frac{1}{R}+\frac{1}{P}}$
- Finden guter Modelle
  - Ansätze: TPR/FPR Graph (ROC), Precision-Recall Graph
  - *Cross-Validation*: Teile wiederholt Daten in Lern- und Validierungsdaten, bestimme jeweils Hypothesen und berechne darauf Metriken (Generalisierung).
    - n-fold-crossvalidation: Teile Daten in n Mengen, trainiere auf n-1 Mengen und teste auf einer.
    - Leave one out: Lerne ohne Beispiel und addiere Fehler für weggelassene Beispiele.

3.4.2 Techniken für Modellfindung

- *Bootstrap*: Ziehe zufällig mit Zurücklegen aus D jeweils $\#|D|$ Beispiele je m Mal.
  Bestimme jeweils Modellparameter, wiederhole und bestimme jeweils Mittelwert und Varianz der Metriken des Modells.
  Analysiere dann Güte/Stabilität und finde Modell mit höherer Güte.
- *Bagging*: Bootstrap Aggregation
  Verwende mehrere Lernmaschinen, ziehe $n < |D|$ Beispiele mit Zurücklegen und bestimme jeweilige Modelle. Kombiniere dann die Lernmaschinen zB mit gewichteter Summe.
  Das resultierende Modell hat eine höhere Güte und kann Abschätzungen über Stabilität machen (große Abweichungen in einzelnen Modellen = Instabil)
- Bagging für Klassifikation
  - Zerlege D in zB D_1, D_2, D_3.
  - Wähle D_1 und bestimme Modell M_1
  - Wähle $D_2 \subset D$ sodass ~50% durch M_1 korrekt geschätzt werden erstelle damit M_2.
  - Wähle $D_3 \subset D$ sodass M_1 und M_2 auf D_3 gegensätzlich schätzen und erstelle damit M_3.
    - Kombiniere die Modelle zu
      $$M = \begin{cases} M_1 & wenn\ M_1 = M_2 \\ M_3 & sonst \end{cases}$$
- *AdaBoost (Adaptive Boosting)*
  Gegeben: Lernmenge D mit $|D| = n$ Beispielen.
  Erstelle iterativ in k Stufen einen komplexen Klassifikator aus k zusammengesetzten Klassifikatoren.
  Ziehe Lernbeispiele entsprechend Gewichtung. Die Gewichtung entspricht dem Klassifikationsergebnis des zuletzt generierten „schwachen“ Klassifikators.
  Das Gewicht von korrekt klassifizierten Beispielen wird verringert, das Gewicht von falsch klassifizierten Beispielen wird erhöht.
  Mit der neuen Lernmenge wird der nächste Klassifikator bestimmt. Die k Klassifikatoren werden zu einem Ensemble zusammengefasst und legen durch gewichteten Mehrheitsentscheid die Klasse eines neuen Beispiels fest.
  - Algorithmus und Bsp siehe Folien F03.28ff

3.5 Lernbarkeit

3.5.1 PAC (Probably Approximate Correct)

- Gegeben:
  - Menge X von Instanzen der Länge n.
  - Konzept C
  - Hypothesenraum H
  - Lerndatenmenge D
- Es kann keine korrekte ($h(x) = c(x) \forall x \in D$) Hypothese gefunden werden, aber eine ε-genaue: $E_D(h) \leq \varepsilon, 0 < \varepsilon < \frac{1}{2}$ („*Approximate Correct*“).
- Diese kann nicht sicher, aber mit Wahrscheinlichkeit $1 - \delta, 0 < \delta < \frac{1}{2}$ („*Probably*“) gefunden werden.
- Das Problem „Finden der Hypothese“ hat polynomialen Aufwand $\sim \frac{1}{\delta}, \frac{1}{\varepsilon}, n$ und Speicheraufwand $\sim C$.
- Die Anzahl der benötigten Lerndaten ist $m \geq \frac{1}{\varepsilon}\left(\ln\left(\frac{1}{\delta}\right) + \ln(|H|)\right)$ („Stichprobenkomplexität“).
  - Gewünschte Sicherheit und Hypothesenraum korrelieren positiv mit benötigten Lerndaten, zulässige Fehler korreliert negativ mit benötigten Lerndaten.

3.6 VC (Vapnik-Chervonenkis) Dimension

Für einen Hypothesenraum $H^\alpha = \{h_\alpha : \alpha \in A\}$ ist die *VC Dimension* $VC(h_\alpha)$ von H^α gleich der maximalen Anzahl Datenpunkten in S, die von H^α beliebig separiert werden können.

Eine Abbildung/Hypothese h separiert die Daten aus S wenn durch h zwei Mengen definiert werden können:

$$\{x|h(x) = 0\}\ \{x|h(x) = 1\}$$

- VC kann als Maß für Datenkomplexität des Lernens genutzt werden.
- Aussagen über Stichprobenkomplexität, Anzahl Lernbeispiele, …
- Vermutung: Je höher VC(h), desto besser kann die Maschine ein Problem einlernen: FALSCH! Overfitting! (Siehe F03.49)

3.7 Abschätzung des Testfehlers

Lernerfolg hängt von Kapazität der Maschine (möglichst gering), Optimierungsmethode und Lernbeispielen ab.

3.8 SRM (Structural Risk Minimization)

Lösungsansatz: Finde Minimum für empirischen Testfehler, also finde $VC(h_\alpha)$ („Maschine“), N („Beispiele“) und α („Minimum des empirischen Fehlers“).

Idealer Metaalgorithmus: Strukturiere Hypothesenraum in Untermengen von Hypothesen, geordnet nach jeweiligem VC-Wert der Dimensionen darin und suche jeweils Optimum mittels empirischen Fehlers. Stoppe wenn Fehler minimal.

Probleme: Berechnung von VC ist schwer und rechenintensiv. Optimierung möglich.

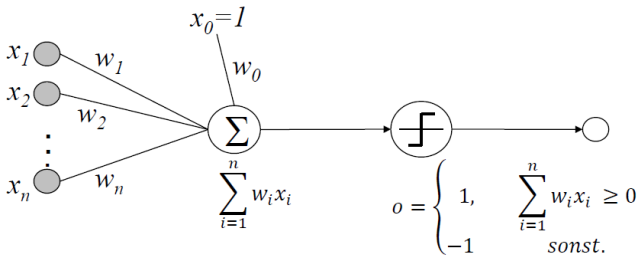
4 Neuronale Netze

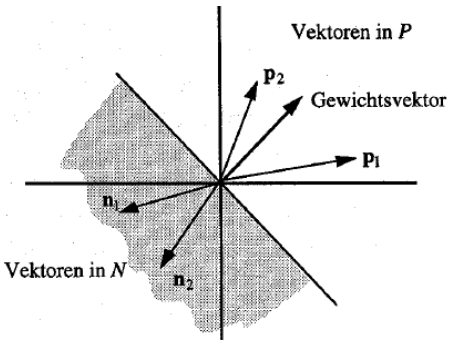
Typische Einsatzfelder: Klassifikation/Mustererkennung, Funktionsapproximation/Regression, Mustervervollständigung.

4.1 Das Perzeptron

Aufbau: Eingabevektor x, Gewichtsvektor w, Summe $\sum$, Aktivierungsfunktion σ (Bsp: $\sigma = \begin{cases} 1, & \sum w_i x_i \geq \varphi \rightarrow 0 \\ -1, & sonst \end{cases}$), Ausgabe des Perzeptrons ist

$$o = \sigma(\sum w_i x_i)$$





In R^2 realisiert das Perzeptron eine Geradentrennung (allgemein eine Trennhyperebene).

Achtung: Es gibt ein Gewicht (w_0) mehr als Eingabeparameter! ($|w| = |x| + 1$)

4.1.1 Lernalgorithmus

```
Gegeben Lerndatenmenge P ∪ N
Erzeuge zufälligen Gewichtsvektor w
Während Zähler < Schwelle:
  Wähle zufälliges Trainingsbeispiel x ∈ P ∪ N
  Wenn x ∈ P:
    Wenn w · x > 0 (korrekt klassifiziert):
      Tue nichts.
    Sonst w · x ≤ 0:
      Setze w := w+x
  Sonst x ∈ N:
    Wenn w · x < 0 (korrekt klassifiziert):
      Tue nichts.
    Sonst w · x ≥ 0:
      Setze w := w-x
  Wenn alle x richtig klassifiziert: break
  Zähler++
```

4.1.2 Vom Perzeptron zum Neuron

$|w| \gg |x|$, durch viele Gewichte ist Anpassung langsam. Lösung: Gradientenabstieg.

Gradientenabstieg: Aktivierungsfunktion sollte differenzierbar und nicht zu 0 differenzierbar sein.

Perzeptronen haben eine niedrige Kapazität, durch Kombinationen vieler Perzeptronen („Neuronen“) sind aber viele Funktionen mgl.

Kernel-Perzeptronen: Einfache Algorithmen wie lineare Trennung können durch „Kernel-Methoden“ realisiert werden, davor Transformation in anderen Raum um auch komplexe Trennung zu ermöglichen.

4.2 Multi Layer Neural Network (MLNN)

- Mehrere versteckte Schichten zwischen Eingabe und Ausgabe. Lernen mittels Backpropagation.
- *Nichtlineare Aktivierungsfunktionen*

Name	$f(x)$	df/dx
Sigmoid	$\frac{1}{1+e^{-x}}$	$f(x)(1-f(x)$
Tangens Hyperbolicus	$\tanh(x)$	$\big(1+f(x)\big)\big(1-f(x)\big)$
ReLU	$\max(0,x)$	$\begin{cases}1, & x>0\\ 0, & sonst\end{cases}$
LeakyReLU	$\begin{cases}x, & x>0\\ \alpha x, & sonst\end{cases}$	$\begin{cases}1, & x>0\\ \alpha, & sonst\end{cases}$

4.2.1 Backpropagation

- Vorgaben: Menge T von Trainingsbeispielen, Lernrate, Netztopologie.
- Ziel: Finde Gewichtsbelegung W, die T wiedergibt.
- Vorgehen: Gradientenabstieg.
- *Algorithmus*
  - Initialisiere Gewichte zufällig mit kleinen Werten
  - Wiederhole bis Abbruchkriterium
    - Wähle Beispielmuster $d \in T$
    - Bestimme Netzausgabe und Fehler bzgl Sollausgabe.
    - Sukzessives Rückpropagieren des Fehlers δ_j.
    - Anpassung der Gewichtsbelegung

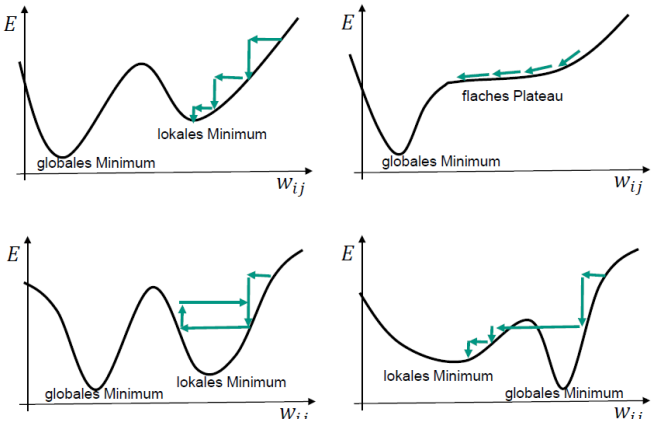
  (Formeln siehe Folien F04.27)
- *Gradientenabstieg*: Allgemeine Delta-Regel
  - Fehlerfunktion: $E_d(w) = \frac{1}{2}\sum_{k\in O}(t_k - o_k)^2$
    - Für Beispiel d, Gewicht w, Zielausgabe $T = \{t_i\}$, tatsächliche Ausgabe $O = \{o_i\}$.
  - Gewichtsänderung nach Gradienten: $\Delta w_{ij} = -\eta \frac{dE_d}{dw_{ij}}$
    - Für Lernrate η, Fehler E_d, Gewicht w_{ij}.
  - Kettenregel $\frac{dE_d}{dw_{ij}} = \frac{dE_d}{dnet_j} \cdot \frac{dnet_j}{dw_{ij}} = \frac{dE_d}{dnet_j} x_{ij}$
    - Für Propagierungsfunktion net_j.
  - Allgemeine Deltaregel: Ableitung des Fehlers für Ausgabeschicht
    - $\frac{dE_d}{dnet_j} = \frac{dE_d}{do_j} \cdot \frac{do_j}{dnet_j}$ (Siehe Folien F04.30)
- *Anpassen der Gewichte*
  - Pattern Learning: Nach jedem Beispiel werden Gewichte angepasst, schnelles Lernen.
  - Mini Batch Learning: Kleine Lernmengen werden mit zurücklegen ausgewählt. Gutes Lernen, SotA.
  - Epochen Learning: Mittelung der Gewichtsänderung über alle Beispiele, Anpassung erst danach. „Echter“ Gradientenabstieg, nicht anfällig für Ausreißer.

  Pattern/ Mini Batch Learning sind nur “Stochastische“ Gradientenabstiege, keine echten.

4.3 Probleme und Optimierungen

4.3.1 Empirischer Fehler - Gradientenabstieg

Lernen ist abhängig von Vorkommen (1) und Ausprägung (3) Lokaler Minima, Steigung der Fehlerfläche (2) und Lernrate (4). Das kann Effekte wie in den jeweiligen Abbildungen verursachen.



- *RPROP (Resilient Propagation):* Normierte Schrittweite und Lernratenanpassung. Lernrate und Gewichtsänderung werden vom Gradientenvergleich abhängig.
  - Beschleunigen auf flachem Plateau, Langsam an Minimum anpassen => Schnelle Konvergenz.
- *Adam (Adaptive Moment Estimation):* Für jeden Parameter werden Lernraten angepasst und jeweils gleitende Schätzer für Mittelwert und Varianz mitgeführt.
- *Xavier-Initialisierung*: Optimierte Initialgewicht-Wahl sodass Vanishing- oder Exploding-Gradients verringert werden.
- Topologieauswahl
  - Theoretische Untersuchung: 1-hidden-Layer macht jede boolsche und kontinuierliche beschränkte Funktion abbildbar. 2-hidden-Layer macht beliebige Funktionen mit beliebiger Genauigkeit abbildbar.

    Also: Geringe Tiefe ist ausreichend.
  - In der Praxis trotzdem verschiedene Ansätze.
  - #Neutronen/Schicht wichtig. Topologie~Kapazität. Viele Neutronen => Overfitting, Wenig Neutronen => Schlechtes Modell.
- Verbesserung der Generalisierung
  - Ziel: Netz trainieren und optimale Topologie finden.

- Ansatz: Großes Netz verkleinern und so Overfitting verhindern.
  - Reduzierungsansatz: Weight Decay, große Gewichte bestrafen durch erweiterte Fehlerfunktion

$$E(w) = E_D(w) + \frac{\lambda}{2}\sum_{ij} w_{ij}^2$$

- Ansatz: Kleines Netz solange erweitert, bis es Daten lernen kann.
  - Meiosis Netzwerke: Bei unsicheren Gewichten werden neue Neuronen eingefügt oder gespalten.
  - Cascade Correlation (siehe unten)

4.3.2 Cascade Correlation

Konstruktives Lernverfahren für MLNN. Algorithmus:

```
Initialisiere 2-schichtiges Netz
Abbruchkriterien: Fehlerschranke, #Neuronen, …
Trainiere: Alle Gewichte werden angepasst
Solange E(w) > Fehlerschranke […]:
  Füge Neuron ein:
    hidden, voll verbunden mit Eingabe/Ausgabe-
    schicht
  Trainiere neues Neuron einmalig
  Trainiere Netz
```

Vereinfachte Version: Kandidat-Neuron wird mit Gewicht -1 zu den anderen Neuronen der gleichen Hidden-Layer verbunden (max unterschiedliche Ausprägung).

- Bewertung:
  - Pro: Je nur eine Ebene wird trainiert, schnelles Lernen, inkrementelles Lernen („weiterlernen“ bei neuen Daten), Iterative Anpassung der Kapazität des Netzes.
  - Contra: Durch spezielle Architektur ist Güte nicht allgemein bestimmbar, Ansatz nicht anwendbar für komplexe Architekturen.

4.3.3 Dropout

Umsetzung von Bagging: k Modelle auf Datenteilmengen mit Zurücklegen trainiert, Ergebnisse mitteln.

Problem: Viele große Netze lernen ist impraktikabel, daher approximative und implizite Umsetzung.

Nehme großes vollständiges Netz und nehme als k Modelle jeweils Subnetze mit maskierten Neuronen, welche jeweils deaktiviert sind. So wird auf allen Subnetzen gleichzeitig gelernt.

4.3.4 Exploding/Vanishing-Gradient Problem

- *Vanishing Gradient*: Bei kleinen Gewichten werden viele kleine Werte multipliziert, das Gradientenabstiegsverfahren lässt Gewichte der unteren Schicht fast unverändert und das Training konvergiert nie zu guter Lsg.
- *Exploding Gradient*: Bei großen Gewichten macht das Lernen instabil und das NN kann nicht aus Daten lernen.

Lösungen: Gradient Clipping (Gradienten während Backpropagation nach oben beschränken), Residual Learning (Skip-Verbindungen über Schichten hinweg => stärkere Rückpropagierung des Fehlers), Early Stopping gegen Overfitting.

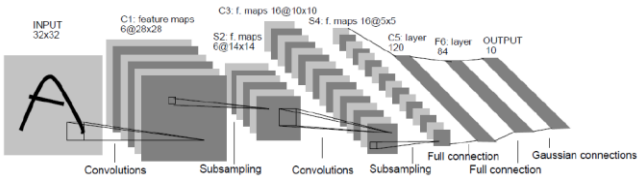
5 Convolutional Neural Networks (CNN)

5.1 Faltung

Siehe Kognitive Systeme, Filtermatrix wird über Bildmatrix geschoben

5.2 Architektur

- Wieso CNNs? Bei großen Eingaben wie Bildern extrem viele Parameter bzw Gewichte.
- Beispiel: LeNet

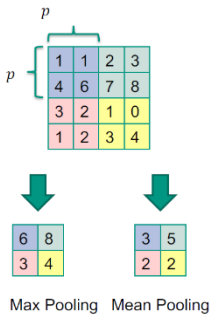


  - 7 Layer: 2 Convolution-, 2 Supersampling/Pooling-, 3 Fully Connected NN Layers.
- Daten werden als Feature Maps zwischen Layern transportiert. Eingabebild ist selbst eine Feature Map, in späteren Stufen können mehrere Feature Maps genutzt werden.

5.3 Layer-Typen und Parameter

5.3.1 Pooling Layer

Zusammenfassung kleiner Bildbereiche.



- Max Pooling: $\max\{a \in A\}$
- Mean Pooling: $\frac{1}{|A|}\sum_{a\in A} a$
- Stochastic Pooling

Nutzen: Lokale Translationsinvarianz, Invarianz gegen leichte Veränderungen und Verzerrungen, Datenreduktion.

5.3.2 Convolution Layer

Wendet Faltungsoperation auf Eingabe der Layer an.

Nimmt r Eingabe-Featuremaps der Größen $m \times m$ und k Filtermatrixen der Größen $n \times n$ entgegen und gibt k Ausgabe-Featuremaps der Größen $m' \times m'$ aus mit

$$m' = \frac{m-n}{s} + 1$$

Meist gilt $q = r$, also genauso viele Eingabemaps wie Filtermaps.

5.3.3 Stride Parameter

Schrittweite der Faltung, muss zur Größe der Eingabemap passen.

Beispiel: 3 x 3 Kernel

Stride = 1

1	0	1	0	0	1	0
0	1	0	0	1	0	0
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	0	0	0	0	0	1

Stride = 2

1	0	1	0	0	1	0
0	1	0	0	1	0	0
0	0	1	0	0	1	0
0	1	0	0	0	0	1
0	0	0	0	0	0	1

Blau = Mittelpunkte des Filter-Kernels

5.3.4 Randbetrachtung

Rand der Eingabe ist bei Faltung ein Problem. Kann ignoriert werden (reduzierte Eingabegröße) oder *gepadded* werden.

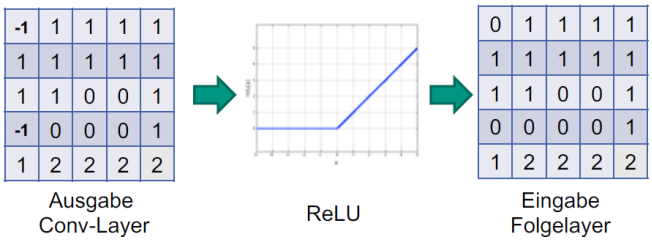
Padding stabilisiert Größenverlauf der Layer. Varianten: *Zero Padding* (mit nullen auffüllen), *Nearest Padding* (Randpixel duplizieren), *Reflext Padding* (Matrix nach außen spiegeln).

Probleme ohne Padding: Informationen am Rand gehen mit der Zeit verloren.

Probleme mit Padding: Am Rand können falsche Strukturen entstehen wie Kanten.

5.3.5 Activation Layer

Layer zur nichtlinearen Aktivierung. Bei CNNs zB durch ReLU (Rectified Linear Unit).



Aktivierungsfunktionen:

- Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$
- Leaky ReLU (LReLU): $f(x) = \begin{cases} 0.01x & wenn \quad x < 0 \\ x & wenn \quad x \geq 0 \end{cases}$
- Parametric ReLU (PReLU): $f(x) = \begin{cases} \alpha x & wenn \quad x < 0 \\ x & wenn \quad x \geq 0 \end{cases}$
- Exponential Linear Unit (ELU):

$$f(x) = \begin{cases} \alpha(e^x - 1) & wenn \quad x < 0 \\ x & wenn \quad x \geq 0 \end{cases}$$

5.4 Lernen über Filter

Feature-Detektoren sind in CNNs in den Filtern der Convolution Layers. *Diese (Gewichte) werden durch Gradientenabstieg gelernt.*

Deconvolutional Networks: Invertierung der einzelnen Schritte zur Visualisierung der Filterlogik für besseres Verständnis der inneren Layers.

5.5 Weight sharing

In CNNs werden nicht alle Gewichte manuell gelernt, sondern nur die in Filtern.

Gewichte werden lokal wiederverwendet (deutlich weniger zu lernende Gewichte, geringere Parametrisierung und dadurch schnelleres Training).

Initialisierung der Gewichte:

- Random Initialization
- Fixed Feature Extractor
  - Gewichte werden aus trainiertem Netz übernommen und die der Featureextraction-Layers werden fixiert.
- Fine-Tuning
  - Gewichte werden aus trainiertem Netz übernommen und auf ausgewählten Schichten mit geringer Lernrate trainiert.
- Pretrained Initialization
  - Gewichte werden aus trainiertem Netz übernommen, aber es wird komplett normal darauf gelernt.

5.6 Wichtige Architekturen

Verschiede Netzarchitekturen, siehe F05.54ff.

Network in Network: Parallele Faltungsoperationen auf Eingabe, Konkatenation der Ergebnisse.

5.7 Fully Convolutional CNN (FCN)

Die Fully Connected Schichten am Ende, die der Klassifikation dienen, werden bei FCNs durch weitere Convulutional Schichten ausgetauscht. Dadurch ist auch die Eingabegröße nicht mehr fix, sondern variabel.

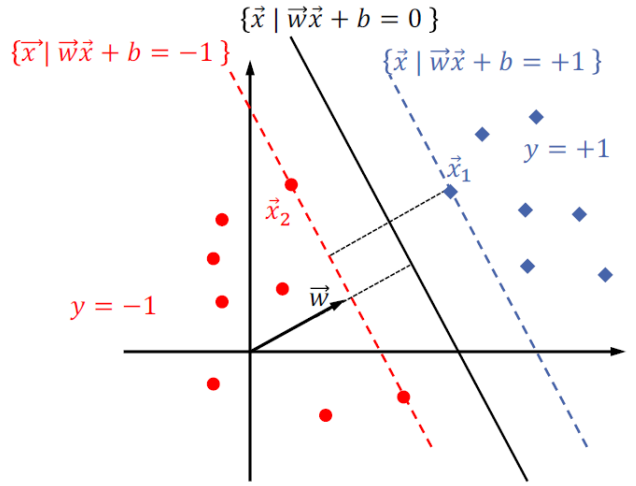
FCN liefert als Ausgabe einen Wahrscheinlichkeitswert pro Klasse. Fully Convolutional Network dagegen: Wahrscheinlichkeits*karte* pro Klasse, die Bereiche zu Klassen zuordnen.

6 Support Vector Machines

6.1 Lineare SVM

Problem: Trenne zwei Mengen von 2D-Punkten durch eine 2D-Linie. Intuition ist, je größer der Rand um die Linie, desto besser wird generalisiert.

Genauer: Finde Hyperebene $\{\vec{x} \in S : \vec{w} \cdot \vec{x} + b = 0, (\vec{w}, b) \in S \times R\}$ mit maximalem Rand.



Rand = Abstand zw. nächsten Punkten

$$\frac{\vec{w}\,(\vec{x}_1 - \vec{x}_2)}{|\vec{w}|}$$

Normierung:

$$\vec{w}\vec{x}_1 + b = +1$$
$$\vec{w}\vec{x}_2 + b = -1$$

Dann gilt:

$$\vec{w}(\vec{x}_1 - \vec{x}_2) = 2$$
$$\frac{\vec{w}}{|\vec{w}|}(\vec{x}_1 - \vec{x}_2) = \frac{2}{|\vec{w}|}$$

Bedinung für die optimale Hyperbene: $\min_{i=1..n} |\vec{w} \cdot \vec{x}_i + b| = 1$ für Lerndaten x_i.

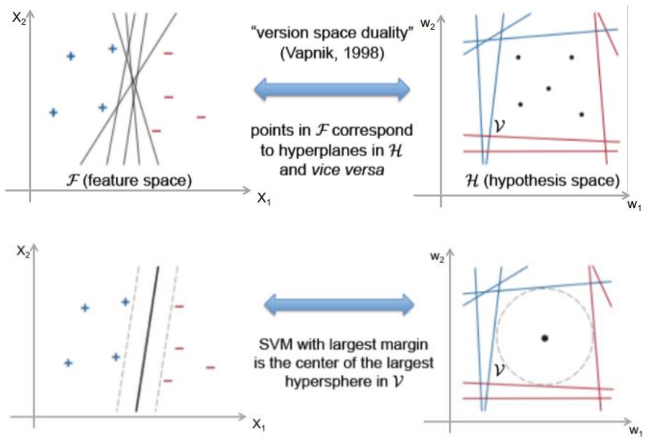
6.2 Generalisierte nichtlineare SVM

Nicht linear trennbare Mengen können durch lineare SVMs nicht diskriminiert werden.

- *Softmargin*: Erlaube geringe Zahl von Missklassifikationen. Höhere Generalisierung, approximiert lineare Diskriminierung falls fast linear diskriminierbar.
- *Nichtlineare Kernelmethoden (SVM):* Für linear nicht trennbare Mengen. Transformiere Daten in einen anderen Raum und löse dort linear mit $\Phi: R^n \to R^m$ mit $m \gg n$.
  - Beispiel: Monom-Transformation $\Phi: R^2 \to R^3, (x_1, x_2) \mapsto (x_1^2, \sqrt{2} \cdot x_1 \cdot x_2, x_2^2)$.
- *Kernel-Trick*, nichtlineare SVM: Da Transformationen und Rechnen auf vielen Dimensionen rechenteuer ist, daher nutze Kernelfunktion $K(\vec{x}, \vec{y}) = \Phi(\vec{x}) \cdot \Phi(\vec{y})$ die das Rechnen gutes Auflösen und so leichtes Rechnen ermöglicht. [Funktioniert, wenn Datennutzung als Skalarprodukt vorkommt, Skalarprodukt wird ersetzt]
  Kernelfunktionen:
  - Skalarprodukt $K(x, y) = x \cdot y$
  - Polinomial (Vovk): $K(x, y) = (x \cdot y + c)^d$
  - Radiale Basisfunktionen (RBF): $K(x, y) = e^{\frac{-||x-y||^2}{2\sigma^2}}$
  - Sigmoid: $K(x, y) = \tanh(\kappa(x \cdot y) + \theta)$

6.3 Versionsraum und strukturelle Risikominimierung (SRM) bei SVM

Merkmals- und Hypothesenraum sind zueinander *dual*. Randbedingung korrekter Klassifikation: $y_i(\vec{w} \cdot \vec{x} + b) \geq 1, i = 1..n$, kann nach $\vec{x}$ im Merkmalsraum oder nach $\vec{w}$ im Hypothesenraum interpretiert werden. Daher wird nach $\vec{w}$ gesucht mit größtem Abstand zu allen Hyperebenen der Datenpunkte.



6.4 Erweiterungen

- *Multiklassen*
  - SVMs sind ähnlich zu CNNs (gelernte Gewichte, Kernel/Aktivierungsneuron, Hiddenschicht, aber nicht Optimierung und theoretischer Hintergrund)
  - Klassifikation auf k Klassen
    - Einer gegen alle: k SVMs, eine pro Klasse, Abstimmung unter SVMs
    - Einer gegen einen: $\frac{k(k-1)}{2}$ SVMs, Abstimmung
    - Mehrfachzugehörigkeit: Wie Einer gegen Alle
    - k-class SVM vom Watkins: Gemeinsames Optimierungsverfahren ohne Abstimmung
- Gewichtete SVM: Minimierungsproblem
- Probabilistische SVM
  - Interpretiere Abstand von Trennhyperebene als Klassifikationswahrscheinlichkeit
- Dichte-Träger Schätzung: Unterscheide kleine Regionen mit den meisten Lernbeispielen (=> f(Region)>0) und Rest (=> f(Rest)<=0).
- Kernel Perzeptron: Folie f06.38ff

6.5 Anwendungen

zB Geschlechts-klassifizierung auf Personenfotos, siehe f06.44ff. Auf niedrig auflösenden Fotos performen SVMs besser als Menschen

6.6 Diskussion

Pro SVMs:

- Optimale Hyperbene führt zu guten Lernergebnissen
- Finden der optimalen VC-Dimension → korrektes Lernen
- Verarbeitung hochdimensionaler Daten → Schnelle Auswertung
- Anwendungsspezifische Kernels (mit Datenverarbeitung)

- Entscheidungen anhand von Randregionen
- Viele Anwendungen: Klassifikation, Regression, PCA
- Probabilistische Sicht: wichtig für semi-überwachtes Lernen
- VS-Beschreibung: gut geeignet für aktives Lernen

Contra SVMs:

- Externe Vorverarbeitung, daher kein „tiefes“ Lernen
- Finden des optimalen Kernels: Aktuelle Forschung
- Parametrisierung des Kernels: Aktuelle Forschung
- Hoher Speicher- und Rechenaufwand, vorallem für Lernen
- Anzahl Vektoren von Problem/Parameter abhängig.

7 Reinforcement Learning (RL)

7.1 Bestandteile des RL Problems

7.1.1 Markov'scher Entscheidungsprozess (MDP)

MDP formalisiert die sequentielle Entscheidungsfindung mit

- S: endliche Menge von Zuständen
- A: endliche Menge von Aktionen
- P: Transitionswahrscheinlichkeitsmatrix
- R: Belohnungsfunktion
  - Belohnung R_t zum Zeitpunkt t beschreibt Güte der jeweiligen Aktion.
- γ: Diskontierungsfaktor $\in [0,1]$

Die meisten Markov Entscheidungsprozesse werden diskontiert weil: Mathematisch pratisch, keine unendlichen Belohnungen durch Zyklen, impliziert Unsicherheit über Zukunft, Verhalten von Tieren/Menschen präferieren sofortige Belohnungen.

7.1.2 Markov Eigenschaft

Ein Zustand erfüllt die Markov Eigenschaft wenn $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$.

7.1.3 Belohnungshypothese

Ziele können als Maximierung der kummulierten Belohnungen beschrieben werden.

7.1.4 Strategie (Policy)

Eine Strategie π beschreibt das Verhalten des Agenten. Es gibt deterministische Strategien $a = \pi(s)$ und stochastische Strategien $a = \pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$.

7.1.5 Model

Ein Model sagt voraus, was die Umgebung als nächstes tut.

- P prognostiziert den nächsten Zustand $P^a_{ss'} = \mathbb{P}[S_{t+1}|S_t = s, A_t = a]$
- R prognostiziert die unmittelbar nächste Belohnung $R^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$

7.1.6 Zustandswertfunktion (State Value Funct.)

Bewertet die Güte eines Zustandes.

$$V_\pi(s) = \mathbb{E}_\pi[\sum_i \gamma^i R_{t+i} \,|S_t = s] \\ = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s]$$

Die Zustandswertfunktion eines MDP ist die erwartete kummulierte Belohnung ausgehend vom Zustand s wenn im folgenden die Strategie π verfolgt wird.

Optimale Zustandswertfunktion: Maximum über allen Strategien: $V^*(s) = \max_\pi V_\pi(s)$

7.1.7 Aktionswertfunktion (Action Value Funct.)

Bewertet die Güte einer Aktion in einem Zustand.

$$Q_{\pi(s,a)} = \mathbb{E}_\pi[\sum_i \gamma^i R_{t+1} \,|S_t = s, A_t = a] \\ = \mathbb{E}_\pi[R_{t+t} + \gamma Q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Die Aktionswertfunktion eines MDP ist die erwartete kummulierte Belohnung ausgehend vom Zustand s und der Aktion a wenn im folgenden die Strategie π verfolgt wird.

Optimale Aktionswertfunktion: Maximum über alle Strategien: $Q^*(s,a) = \max_\pi Q_\pi(s,a)$

Ein MDP ist durch finden der optimalen Zustands- und Aktionswertfunktionen gelöst.

7.1.8 Bellman Optimalitätsgleichungen

$$V^*(s) = \max_a R^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'} V^*(s')$$

$$Q^*(s,a) = \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'} \max_{a'} Q^*(s', a')$$

7.2 Grundlegende Lösungsverfahren

7.2.1 Dynamische Programmierung

Policy Iteration

$\hat{\pi}^*$ Schätzung von π^*

Initialisiere $V(s)$ und $\hat{\pi}^*(s) \in A(s)\ \forall\, s \in S$ zufällig
Wiederhole
  $\hat{\pi}^* \leftarrow \hat{\pi}^{*'}$
  Für alle $s \in S$
    $V_{\hat{\pi}^*}(s) \leftarrow \mathcal{R}_s^{\hat{\pi}^*(s)} + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^{\hat{\pi}^*(s)} V_{\hat{\pi}^*}(s')$ — Evaluierung Strategie
    $\hat{\pi}^{*'}(s) \leftarrow \arg\max_a \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'} V_{\hat{\pi}^*}(s')$ — Verbesserung Strategie
Bis $\hat{\pi}^* = \hat{\pi}^{*'}$

Value Iteration

$\hat{V}^*$ Schätzung von V^*

Initialisiere $\hat{V}^*(s)\ \forall\, s \in S$ zufällig
Wiederhole
  $\hat{V}^*(s) \leftarrow \hat{V}^{*'}(s)$
  Für alle $s \in S$
    Für alle $a \in A$
      $Q(s,a) \leftarrow \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'} \hat{V}^*(s')$ — Evaluierung Aktionswertfunktion
    $\hat{V}^*(s) \leftarrow \max_a Q(s,a)$ — Verbesserung Zustandswertfunktion
Bis $\hat{V}^*(s) = \hat{V}^{*'}(s)$
$\hat{\pi}^*(s) = \arg\max_a \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_{ss'} \hat{V}^*(s')$

Beispiele siehe Folien F07.18ff

7.2.2 Monte Carlo

- Lernen direkt aus Erfahrungsepisoden
- Modellfrei, benötigen kein Wissen über Übergänge/Belohnungen
- Lernen aus kompletten Episoden, kein Bootstrapping
- Intuitive Vorgehensweise: Wert = ØBelohnung
- Kann aber nur auf episodische MDPs angewendet werden. Alle Episoden müssen terminieren.

$\hat{\pi}^*$ Schätzung von π^*

Initialisiere $\hat{V}^*(s)\ \forall\ s \in S$ zufällig
Wiederhole:
Generiere Episode mit $\hat{\pi}^*$ von zufälligem Zustand mit zufälliger Aktion
Für alle Tupel s, a der Episode:
$G \leftarrow$ Entlohnung welche auf s, a folgt
Füge G zu $Entlohnungen(s, a)$ hinzu
$Q(s, a) \leftarrow Durchschnitt(Entlohnungen(s, a))$ Evaluierung Aktionswertfunktion
Für jedes s der Episode:
$\hat{\pi}^*(s) \leftarrow \arg\max_a Q(s, a)$ Verbesserung Strategie

7.2.3 Temporal Difference Learning

- Lernt direkt aus Erfahrungsepisoden
- Modellfrei
- Lernt aus unvollständigen Episoden mit Bootstrapping
- Aktualisiert eine Schätzung mit neuer Schätzung

$\hat{V}_\pi$ Schätzung von V_π
π Strategie welche evaluiert werden soll
α Lernrate

Wiederhole:
Generiere Episode mit π
Für alle Tupel s, a, R der Episode:
$\hat{V}_\pi(s) \leftarrow \hat{V}_\pi(s) + \alpha[R + \gamma \hat{V}_\pi(s') - \hat{V}_\pi(s)]$ Aktualisierung Zustandswertfunktion

7.2.4 Bootstrapping und Sampling

Bootstrapping (Shallow Backups): Aktualisierung schätzen: Dynamische Programmierung, Temproal Difference Learning

Sampling (Sample Backups): Aktualisierung aufgrund Erwartungswert: Monte Carlo, Temporal Difference Learning

7.2.5 Taxonomie

- Bewertungsbasiert: Keine Strategie, Bewertungsfkt
- Strategiebasiert: Strategie ohne Bewertungsfkt
- Actor Critic: Strategie und Bewertungsfkt

7.3 Reinforcement Learning Algorithmen

7.3.1 SARSA

State, Action, Reward, State, Action

Initialisiere $Q(s, a)\ \forall\ s \in S, a \in A(s)$ zufällig

Wiederhole (für jede Episode):
Initialisiere s
Wähle a von s mit Strategie abgeleitet von Q (z.B. ϵ-greedy)
Wiederhole (für jeden Schritt der Episode):
Nimm Aktion a, beobachte R, s'
Wähle a' von s' mit Strategie abgeleitet von Q (z.B. ϵ-greedy)
$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma Q(s', a') - Q(s, a)]$

7.3.2 Q-Learning

Initialisiere $Q(s, a)\ \forall\ s \in S, a \in A(s)$ zufällig

Wiederhole (für jede Episode):
Initialisiere s
Wiederhole (für jeden Schritt der Episode):
Wähle a von s mit Strategie abgeleitet von Q (z.B. ϵ-greedy)
Nimm Aktion a, beobachte r, s'

$$Q(\mathrm{s}, \mathrm{a}) \leftarrow Q(\mathrm{s}, \mathrm{a}) + \alpha\left[R + \gamma \max_{a'} Q(\mathrm{s}', \mathrm{a}') - Q(\mathrm{s}, \mathrm{a})\right]$$

7.3.3 Policy Gradient

Initialisiere Strategieparameter $\boldsymbol{\theta} \in R^d$ zufällig → $\pi(a|s, \boldsymbol{\theta})$

Wiederhole:
Generiere eine Episode $S_0, A_0, R_1, \ldots S_{T-1}, A_{T-1}, R_T$ mit $\pi(\cdot|\cdot, \boldsymbol{\theta})$
Für jeden Schritt der Episode:
$G \leftarrow$ Entlohnung von Schritt t
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[\gamma G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta}]$

7.4 Sonstige Informationen

7.4.1 Exploration vs Exploitation

Reinforcement Learning lernt nach Trial-and-Error. Änderungen der Suchstrategie von global (Exploration) zu lokale (Exploitation) während des Lernprozesses.

7.4.2 On-Policy Learning/ Off-Policy Learning

- *On-Policy*: Evaluiere oder verbessere Strategie, welche zur Entscheidungsfindung verwendet wird.
  - Learning on the Job.
- *Off-Policy*: Evaluiere oder verbessere Strategie, welche sich von Strategie zur Entscheidungsfindung unterscheidet.
  - Look over someone's shoulder.

7.4.3 Vorwärtssicht TD(γ)

Lernen von langen Aktionssequenzen kann wichtig sein, nicht nur einzelne Aktionen. Vorwärtssicht schaut in die Zukunft um die Entlohnung zu berechnen. Nur für terminierte Episoden.

Dagegen Rückwärtssicht $TD(\gamma)$: Etstellt Verantwortlichkeitsspur für jeden Zustand.

7.4.4 Lernen und Planen

- Reinforcement Learning
  - Unbekannte Umgebung, Agent interagiert mit Umgebung und verbessert Strategie
- Planen

- Modell der Umgebung ist bekannt, Agent führt mit seinem modell Berechnungen durch und verbessert Strategie.

8 Entscheidungsbäume

8.1 Motivation

Motivation: Partitioniere Raum so, dass Daten in gleicher Region gleiche Klasse haben.

Geeignete Problemstellungen:

- Instanzen lassen sich als Attribut-Wert Paar beschreiben
- Zielfunktion besitzen diskrete Ausgabewerte
- Disjunkte Hypothesen
- Potentiell noisy Beispieldaten
- Potentiuell fehlende Attributwerte auf Beispieldaten

8.2 ID3

Knoten = Wurzel
Schleife:
1. A ← Das beste Entscheidungsattribut für den nächsten Knoten.
2. Weise A als Entscheidungsattribut für den nächsten Knoten zu.
3. Füge für jeden möglichen Wert von A einen Nachfolgeknoten ein.
4. Verteile die Trainingsdaten gemäß ihrer Attributwerte auf die Nachfolgeknoten.
5. **Wenn** die Trainingsdaten perfekt abgebildet (klassifiziert) sind:
   Break
   sonst iteriere über die Nachfolgeknoten.

8.2.1 Einordnung

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	*symbolisch* vs subsymbolisch
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

8.2.2 Auswahl des besten Testattributs

Entropie beschreibt Homogenität der Testdaten

$$Entropie(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

(S=Menge der Trainingsbeispiele, p_+ Anteil der positiven Beispiele in S, p_- analog negative Beispiele)

Informationsgewinn: Erwartete Reduzierung der Entropie durch Einsortieren über Attribut A:

$$Gewinn(S, A) = Entropie(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropie(S_v)$$

(V(A)=Menge aller mgl Attributwerte von A, $S_v \subset S$ für die A den Wert v annimmt)

Maximierung des Gewinns führt zu Baum mit niedriger Tiefe.

Rechenbeispiele siehe Folien F08.15.

8.2.3 Präferenzen-/Restriktionsbias

- Präferenzenbias: Ordnung auf dem Raum der Hypothesen, wähle Hypothese h mit höchster Präferenz.
- Restriktionsbias: Schränke Hypothesenraum zB mit linearer Schwellwertfunktion ein.

8.2.4 Occam's Razor

Bevorzuge kurze Hypothesen, denn: Es gibt weniger kurze als lange Hypothesen, kurze korrekte Hpypothesen sind vermutlich kein Zufall, lange korrekte Hypothese vielleicht schon. Außerdem effizienter.

8.2.5 Overfitting

Eine Hypothese h overfittet die Daten D, wenn es eine alternative Hypothese h' gibt mit $Fehler_T(h) < Fehler_T(h')$ und $Fehler_D(h) > Fehler(h')$ auf Trainingsdaten T und gesamter Datenverteilung D.

Vermeiden: Frühzeitiger Baumwachsumsstop oder Pruning.

Optimale Baumlänge mittels seperater Testdaten, statistischem Test auf Trainingsdaten oder Maß für Kodierungskomplexität (MDL).

Reduced Error Pruning: Teile Daten in Trainings- und Testdaten. Solange Pruning keine negativen Effekte hat, finde den Knoten, dessen Entfernen die beste Auswirkung auf die Klassifikationsrate bzgl. Testdaten hat.

Reduced Error Pruning erhöht Fehler bei wenig Daten aber noch mehr!

8.2.6 Erweiterungen

8.2.6.1 Attribute mit vielen Werten

Gewinn(S,A) bevorzugt Attribute mit vielen Werten. Lösung: Bestrafe Attribute

$$GewinnAnteil(S, A) = \frac{Gewinn(S, A)}{SlittInformationen(S, A)}$$

$$SplitInformationen(S, A) = -\sum_{v \in Values(A)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

mit $S_v \subset S$ Trainingsdaten mit Wert v bei Attribut A.

8.2.6.2 Kontinuierliche Attributwerte

Vorgehen: Dynamisch gewählter neues diskretes Attribut $A_c := A > c$. Wahl des Schwellwerts c nach Informationsgewinn, sortiere Beispiel nach Werten und wähle Wert zwischen Instanzen mit verschiedenen Klassen.

Rechenbeispiel siehe Folie F07.32.

8.2.6.3 Unbekannte Attributwerte

Wähle als Wert den häufigsten Attributwert der Beispiele, häufigsten Wert der Beispiele derselben Klasse oder nach Wahrscheinlichkeitsvariable gewählten Wert.

8.2.6.4 Attribute mit Kosten

Finde korrekten Entscheidungsbaum mit niedrigen erwarteten Kosten mit $\frac{Gewinn(S,A)^2}{Kosten(A)}$.

8.2.6.5 Große Datenmengen

Window, trainiere Baum mit Teildatenmenge und klassifiziere Rest damit. Retrainiere mit falsch klassifizierten Instanzen.

8.3 C4.5

Weiterentwicklung von ID3. Unterstützt kontinuierliche und fehlende Attributwerte. Nutzt Rule Post-Pruning.

8.3.1 Einordnung

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	*symbolisch* vs subsymbolisch
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

8.3.2 Rule Post-Pruning

1. Erstelle potentiell overfitteten Baum aus Trainingsdaten
2. Konvertiere Baum in IF-THEN Regeln (IF=Attributtests, THEN=Klasse)
3. Prune/Generalisiere Regeln so lange sich Vorhersagegenauigkeit nicht verschlechtert
4. Sortiere Regeln nach Klassifikationsgüte und nutze sie in dieser Reihenfolge.

Nutze eher pessimistische Abschätzung der Güte der Regeln.

Vorteile:

- Unterscheidung von Kontexten, in denen Entscheidungsknoten benutzt werden (Eine Regel pro Pfad im Baum)
- Keine Unterscheidung von Wurzelnahen und Blattnahen Attributen, dadurch einfacheres Prunen.
- Gute Lesbarkeit.

8.4 ID5R

Inkrementeller Aufbau des Baums, äquivalent zu ID3.

Blätter bezeichnen Klassen, Entscheidungsknoten zweigen nach Attributwerten b und zählen jeweils positive und negative Beispiele sowie noch ausstehende Attributtests.

Algorithmen siehe F08.47ff

8.4.1 Einordnung

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	*symbolisch* vs subsymbolisch
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	*inkrementell* vs nicht inkr.
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

8.5 Random Forests

Mehrere Entscheidungsbäume. Jeder Baum klassifiziert, die Klasse mit den meisten Entscheidungen wird gewählt.

Schnelles Training (da kleinere Bäume), parallelisierbar, effizient für große Datenmengen.

Kein Pruning, Overfitting erlaubt. Bäume sollten unkorelliert sein.

9 Hidden Markov Modelle

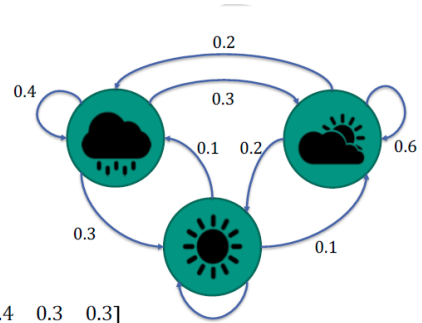
9.1 Motivation

Reale Prozesse sind nichtdeterministisch, aber stochastisch beschreibbar. Reale Signale sind meist noisy und besitzen nichtdeterministische Eigenschaften.

9.2 Diskreter Markov-Prozess

Diskreter Markov Prozess: N diskrete Zustände S mit Zeitpunkten der Zustandsübergänge $t \in [1, N]$ und aktuellem Zustand q_t zum Zeitpunk t.

$\pi_i = \mathbb{P}(q_1 = S_i)$ beschreibt Anfangszustandswahrscheinlichkeiten, Übergangswahrscheinlichkeits-Matrix $a_{ij} = \mathbb{P}(q_t = S_j | q_{t-1} = S_i) = \mathbb{P}(S_j | S_i)$. (Wie MDP-Übergangsmodell ohne Aktionen)



$$A = (a_{ij}) = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$

Rechenbeispiel siehe F09.6.

Markov-Bedingung:

- Beschränkter Horizont: Die Wsk einen Zustand zu erreichen ist nur von seinem direkten Vorgängerzustand abhängig.
- Zeitinvarianz: Die Wsk eines Zustandsüberganges ist unabhängig vom Zeitpunkt.

9.3 Hidden Markov Modelle (HMMs)

Bei HMMs können Zustände nur indirekt beobachtet werden, Beobachtung ist eine stochastische Funktion des Zustands.

HMMs beschreiben einen doppelt stochastischen Prozess: Der zugrunde liegende stochastische Prozess kann nur indirekt durch eine andere Menge von stochastischen Prozessen beobachtet werden, die eine Beobachtungssequenz produziert.

- Definition: Ein HMM ist $\lambda = \{S, V, A, \Pi\}$ mit
  - S: Menge von N Zuständen S_i
  - V: Menge von Ausgaben/Beobachtungen V_i
  - $A = (a_{ij}) \in [0,1]^{N \times N}$: Matrix der Übergangswsks
  - $B = (b_{ik}) \in [0,1]^{N \times M}$: Matrix der Ausgabewsks
  - $\Pi = \{\pi_i | \pi_i = \mathbb{P}(q_1 = S_i)\}$: Verteilung des Anfangszustandes
- Es gilt
  - Zustandswsk: $\mathbb{P}(q_{t+1} = S_j) = \sum_{S_j \in S} a_{ij} \cdot \mathbb{P}(q_t = S_i)$
  - Ausgabewsk: $\mathbb{P}(o_t = V_k) = \sum_{S_j \in S} b_{ik} \cdot \mathbb{P}(q_t = S_i)$

Wobei q_t den Zustand und o_t die Ausgabe zum Zeitpunkt t bezeichnet.

Beispiel siehe F09.10ff.

9.4 3 grundlegenden Probleme

- *Evaluationsproblem*
  - Für ein HMM, wie hoch ist die Ausgabewsk $P(O|\lambda)$ für die Ausgabe O?
  - Also: Wie gut erklärt ein Modell eine Beobachtungssequenz?
  - Beispiel: Erkenne unbekannte Wörter durch Finden des besten Modells.
- *Dekodierungsproblem*
  - Für ein HMM, welche ist die wahrscheinlichste Zustandsfolge Q, die O erklärt?
  - Also: Finde die korrekte (verborgene) Zustandssequenz
  - Beispiel: Verstehe aufgebaute Modelle und verbessere damit.
- *Lern- oder Optimierungsproblem*
  - Für Ausgabesequenzen O und ein Modell-Suchraum, passe HMM an sodass O besser erklärt wird.
  - Also: Training
  - Beispiel: Lernen HMM ein für jedes zu erkennende Wort.

9.5 Lösungen der Probleme

9.5.1 Evaluationsproblem

Nativer Ansatz:

$$\mathbb{P}(O|\lambda) = \sum_{\forall Q} \mathbb{P}(O, Q|\lambda)$$

mit $\mathbb{P}(O, Q|\lambda) = \mathbb{P}(O|Q, \lambda)\mathbb{P}(Q|\lambda)$ und $\mathbb{P}(O|Q, \lambda) = \Pi_{t=1}^{T} \mathbb{P}(o_t|q_t, \lambda)$ und $\mathbb{P}(Q|\lambda)$ =Wsk der Zustandsfolge $q_1, q_2, \dots, q_T$. Aufwand $O(2T \cdot N^T)$.

Lösung: Teilresultate zwischenspeichern.

9.5.1.1 Vorwärts-Algorithmus

1. Initialisierung: $\alpha_1 = \pi_{S_i} b_{S_i}(o_1)\ \forall i \in [1, N]$
2. Induktion: $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1})$
   $\forall t \in [1, T-1]\ \forall j \in [1, N]$
3. Terminierung: $\mathbb{P}(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

b_i beschreibt Ausgabewsks des Zeichens in Zustand i, a_{ij} beschreibt Übergangswsk von i nach j.

9.5.1.2 Rückwärts-Algorithmus

1. Initialisierung: $\beta_T(i) = 1\ \ \forall i \in [1, N]$
2. Induktion: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$
   $\forall t \in [1, T-1]\ \forall i \in [1, N]$
3. Terminierung: $\mathbb{P}(O|\lambda) = \sum_{j=1}^{N} \pi_j b_j(o_1) \beta_1(j)$

9.5.2 Dekodierungsproblem

Ansatz: Teste mit Vorwärts/Rückwärts und finde so die wahrscheinlichsten Zustandsfolgen.

Viterbi-Algorithmus: Rekursive Maximierung entlang möglicher Zustandsfolgen.

- Wie Vorwärts, aber statt Summe wird max-Operator angewendet und der beste Vorgängerzustand $\psi_{t+1}(j)$ gemerkt.

1. Initialisierung: $v_1(i) = \pi_{S_i} b_{S_i}(o_1)\ \forall i \in [1, N]$,
   $\psi_1(i) = 0$
2. $\forall S_j \in S, \forall j \in [1, N], \forall t \in [2, T]$:
   a. $v_t(S_j) = b_j(o_t) \cdot \max_{1 \le i \le N} [v_{t-1}(S_i) \cdot a_{ij}]$
   b. $\psi_t(j) = \arg\max_{1 \le i \le N} [v_{t-1}(S_i) \cdot a_{ij}]$
3. Maximale Güte: $v^* = \max_{1 \le i \le N} [v_T(S_i)]$
4. Endknoten des besten Pfades: $q_T^* = \arg\max_{1 \le i \le N} [v_T(S_i)]$
5. Viterbi-Pfad rückwärts bestimmen: $q_t^* = \psi_{t+1}(q_{t+1}^*)$ für t=T-1, T-2, …, 1

9.5.3 Lern-/Optimierungsproblem

Kein optimaler Weg, aber lokale Maximierung von $\mathbb{P}(O|\lambda)$ mit iterativem Ansatz wie Baum-Welch möglich.

Suche $\bar{\lambda} = \arg\max_{\lambda} \mathbb{P}(O|\lambda)$

Ansatz: Vorgegebene #Zustände, variiere A, B, Π.

Algorithmus (Expectation Maximization):

1. Nehme zufälliges λ, berechne $\mathbb{P}(O_{training}|\lambda)$
2. While
   a. E: Bestimme erwartete Anzahl von Zustandsübergängen und Symbolausgängen
   b. M: Schätze Parameter neu
   c. Wenn lokales Maximum erreicht, break

Dazu: $\xi_t(i,j)$ = Wsk eines Zustandsübergangs von i nach j zur Zeit t. $\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$ = Wsk, zur Zeit t in Zustand i zu sein.

9.6 Eigenschaften

Arten von HMMs: Ergodisches Modell (jeder Zustand kann von jedem anderen in endlich vielen Schritten erreicht werden) oder Links-nach-rechts Modell (Zustandsindex wird mit der Zeit größer oder bleibt gleich).

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	symbolisch vs *subsymbolisch*
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

9.7 Anwendungsbeispiele

ZB Spracherkennung, Gestenerkennung, Ampelzustandsschätzung. Immer wenn stochastische Prozesse nur indirekt erfasst werden können.

10 Lernen nach Bayes

10.1 Einordnung Bayes-Methoden

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	symbolisch vs *subsymbolisch*
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

10.2 Bedingte Unabhängigkeit

Zufallsvariable X ist bedingt unabhängig von Y gegeben Z, wenn

$$\mathbb{P}(X|Y,Z) = \mathbb{P}(X|Z)$$

10.3 Motivation, Theorem nach Bayes

Kombiniere vorhandenes Wissen (a-priori Wsks) mit beobachteten Daten.

Nutzten zB zur Analyse anderer Verfahren, Bayes als „Gold-Standard".

Wahrscheinlichkeitstheorie Regeln:

- $\mathbb{P}(A \wedge B) = \mathbb{P}(A|B)\mathbb{P}(B)$
- $\mathbb{P}(A \vee B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \wedge B)$
- Theorem der total Wsk: $\mathbb{P}(B) = \sum_{i=1}^{n} \mathbb{P}(B|A_i)\mathbb{P}(A_i)$ für sich gegenweise ausschließende Ereignisse A_i mit $\sum A_i = 1$.
- Satz von Bayes: $\mathbb{P}(h|D) = \frac{\mathbb{P}(D|h)\mathbb{P}(h)}{\mathbb{P}(D)}$
  - $\mathbb{P}(h|D)$ ist a-posteriori Wsk, also dass Hypothese h unter Daten D gilt
  - $\mathbb{P}(D|h)$ ist die Likelyhood, also Wsk dass D unter h auftritt
  - $\mathbb{P}(h)$ ist a-priori Wsk, Wsk dass h aus H gültig ist
  - $\mathbb{P}(D)$ ist Wsk des Auftretens der Daten unabhängig der Hypothese

10.4 MAP-/ ML-Hypothesen

- Maximum a-posteriori (MAP) Hypothese: $h \in H$ mit größter Wsk auf beobachteten Daten D.
  - $h_{MAP} = \arg\max_{h \in H} \mathbb{P}(h|D)$
    $= \arg\max_{h \in H} \frac{\mathbb{P}(D|h)P(h)}{P(D)}$ (Bayes)
    $= \arg\max_{h \in H} \mathbb{P}(D|h)P(h)$ (P(D) ist konstant)
  - Max Likelyhood: $h_{ML} = \arg\max_{h_i \in H} P(D|h_i)$ (Alle Hypothesen gleich wahrscheinlich)
  - Brute Force: Berechne a-posteriori Wsk $P(h|D)$ für alle $h \in H$, nehme h_{MAP}, also h mit höchster a-posteriori Wsk.
    - Beispiel siehe Folien F10.12ff
- Konzeptlernen: Nicht verrauschte Trainingsdaten mit gleich wahrscheinlichen Hypothesen. Liefert Suche vom Speziellen zum Allgemeinen eine MAP-Hypothese?
  - $\mathbb{P}(D|h) = 1 \Leftrightarrow$ h erkennt Trainingsdaten exakt, 0 sonst. $\mathbb{P}(h) = \frac{1}{|H|}$.
  - Unter den Vorraussetzungen gibt jeder konsistenter Lerner eine MAP-Hypothese aus.
  - Ein *konstistenter Lerner* ist ein Verfahren, das eine Hypothese liefert, welche keine Fehler auf Trainingsdaten macht.

10.5 Optimaler Bayes-Klassifikator

Gegeben einer Hypothese, wie klassifiziert man nun damit? Vorsicht, Ausgabe von $h_{MAP}(x)$ ist *nicht* die wahrscheinlichste Klassifikation!

Optimale Klassifikation nach Bayes:

$$v_{OB} = \arg\max_{v_j \in V} \sum_{h_i \in H} \mathbb{P}(v_j|h_i)\mathbb{P}(h_i|D)$$

für Klassen V und Trainingsdaten D. Rechenbeispiel F10.22.

Vorteil: Bestes Klassifikationsverfahren bei gleichem Bedingungen.

Nachteil: Sehr kostenintensiv bei vielen Hypothesen.

Alternativen: Gibbs-Algorithmus (Ensemble mit zufälliger h-Wahl) oder Voting-Gibbs.

10.6 Naiver Bayes-Klassifikator

Gegeben: Instanz x aus bedingt unabhägnigen Attributen a_i, endlich Menge Klassen V, Menge klassifizierter Beispiele.

Naiver Bayes Klassifikator:

$$v_{NB} = \arg\max_{v_j \in V} \mathbb{P}(v_j) \prod_i \mathbb{P}(a_i|v_j)$$

$\mathbb{P}(v_j)$ und $\mathbb{P}(a_i|v_j)$ wird nach Häufigkeiten in D geschätzt. Wenn Attribute bedingt unabhängig sind, ist Ergebnis äquivalent zu MAP-Klassifikation. Aber schneller, da keine explizite Suche im Hypothesenraum!

Beispiel F10.28

Beispielrechnung:

$$\mathbb{P}(Ja) \cdot \mathbb{P}(kalt|Ja) \cdot \mathbb{P}(sonnig|Ja) \cdot \mathbb{P}(wind|Ja)$$

10.6.1 Schätzen von Attribut-Wsks

Problem: Klasse kann konkretes Attribut in Testdaten vlt gar nicht annehmen. $\mathbb{P}(a_i|v_j) = 0 \Rightarrow Wsk = 0$

Lösung: m-Laplace-Schätzer.

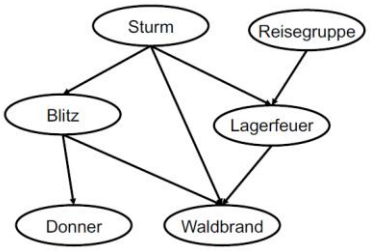
10.7 Bsp: Textklassifikation

Siehe Folien F10.30ff

10.8 Bayessche Netze

Motivation: Naive Bayes-Annahme (Attribute sind bedingt unabhängig) oft zu restriktiv.

Bayessche Netz: Gerichteter azyklisher Graph. Jede Zufallsvariable ist ein Knoten. Kante garantieren, dass Zufallsvariable von ihren Nicht-Nachfolgern bedingt unabhängig ist, gegeben ihren direkten Vorgängern. X ist Nachfolger von Y, wenn ein gerichteteer Pfad von Y nach X existiert.



$P(L|S, R)$

	S, R	$S,\neg R$	$\neg S,R$	$\neg S,\neg R$
L	0.4	0.1	0.8	0.2
$\neg L$	0.6	0.9	0.2	0.8

Lagerfeuer

zB: Lagerfeuer ist von Donner, Blitz, Sturm und Reisegruppe bedingt unabhängig gegeben Reisegruppe und Sturm.

Es gilt: $\mathbb{P}(y_1, \dots, y_n) = \Pi_{i=1}^{n} \mathbb{P}(y_i|DirekteVorgänger(Y_i))$

Inferenz von Netzvariablen: zB Approximation durch Monte Carlo, oder exakte Inferenz durch spezielle Netztopologien.

Lernen:

- Struktur bekannt, alle Variablen beobachtbar: Wie naiver Bayes Klassifikator
- Struktur bekannt, nicht alle Variablen beobachtbar: Gradientenabstieg, EM
- Struktur unbekannt: Heuristik

10.9 EM-Algorithmus

Expectation Maximization.

- Problemstellung: Daten nur partiell beobachtbar, Verteilung parametrisch modellierbar, unüberwachtes Clustering, Überwachtes Lernen. Hypothesenparameter sollen geschätzt werden.
- Anwendungen: Bayessche Netze trainieren, HMMs trainieren
- Ansatz: Wechsle Iterativ zwischen E- (Schätzen der nichtbeobachtbaren Werte) und M-Schritt (Parameteranpassung)
- Werte werden aus einer zufällig gezogenen von k Gauß-verteilungen generiert. Suche Maximum Likelyhood Schätzung für Gauß-Mittelwerte μ_i
- Beispiel: k=2 Gauß-Mittelwerte berechnen
  - Initialisierung: Wähle Mittelwerte μ_1, μ_2 zufällig
  - E-Schritt: berechne Erwartungswert $E(z_{ij})$ für alle versteckten Variablen Z_{ij}
  - M-Schritt: Berechne neue Max-Likelyhood Hypothese mit neuen Mittelpunkten:
    - $\mu'_j = \frac{1}{m}\sum_{i=1}^{m} E[z_{ij}]x_i$
  - Iteriere.

EM konvergiert gegen eine lokale Max-Likelyhood hypothese und liefert Schätzungen für versteckte Variablen. Sucht die Hypothese, die Erwartungswert der Wsk der gesamten (teils unbekannten) Daten unter der gesuchten Hypothese maximiert.

10.9.1 Allgemeines Verfahren

- Likelyhood Funktion $Q(h'|h)$ berechnet Wsk von $Y = X \cup Z$ unter Verwendung der beobachtbaren Daten X und dem aktuellen Parameter von h, um Z zu schätzen.
- E-Schritt: Berechne $P(Y|X, h)$
- M-Schritt: Ersetze Hypothese durch neue, welche Q maximiert.

11 Instanzbasiertes Lernen

11.1 Einführung

- Lazy Learning
- Effizientes Training, komplexere Klassifikation
- Lokale Zielfunktionsapproximation für jede Anfrage
- Bewertung:

- Positiv: Komplexe Zielfunktionen darstellbar, Informationen aus Trainingsdaten gehen nicht verloren
- Negativ: Komplexe Neuklassifikationen, Problem irrelevanter Eigenschaften

11.2 k-NN Algorithmus

Instanzen mit n Attributen auf n-dimensionalen Raum, Unterschiede durch euklidische Distanz.

$f: R^n \to V$ wird mit Trainingsbeispielen $(R^n \times V)^{|D|}$ gelernt.

Training: Trainingsbeispiele einfach merken.

Klassifikation: $f(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, c(x_i))$ für die k nächsten Nachbarn zu x_q und $\delta(a,b) = 1 \Leftrightarrow a = b$. Also: Finde Klasse, sodass von den k nächsten Nachbarn die meisten dieselbe Klasse haben.

- Normalisierung der Eingabevektoren sinnvoll, da bei stark ungleichen Eingabedimensionen stark verzerrt wird (zB Attribut Alter 0-100 und Gehalt 0-100k)
- Abstandsgewichteter k-NN:

$$f(x_q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, c(x_i))$$

  mit $w_i = \frac{1}{d(x_q, x_i)^2}$, sodass nahe Nachbarn stärker gewichtet werden.

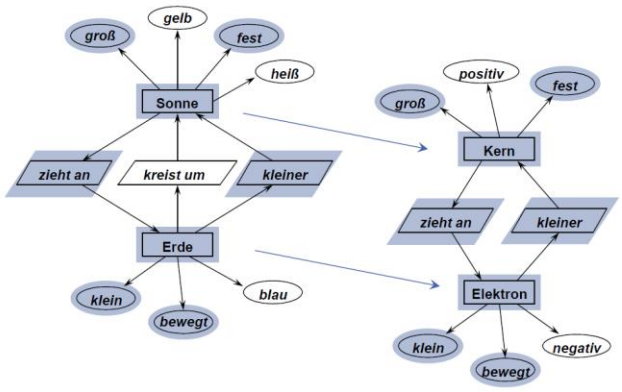
- Bewertung:
  - Induktiver Bias: Klassifikation einer Instanz ähnlich zu ähnlichen Instanzen
  - Positiv: Robust auf verrauschte Trainingsdaten
  - Negativ: Curse of Dimensionality (viele Attribute vermutlich unwichtig), Speicheroranisation, Hauptaufwand in Klassifikation statt Training

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	symbolisch vs *subsymbolisch*
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

11.3 Case-Based Reasoning

11.3.1 Motivation

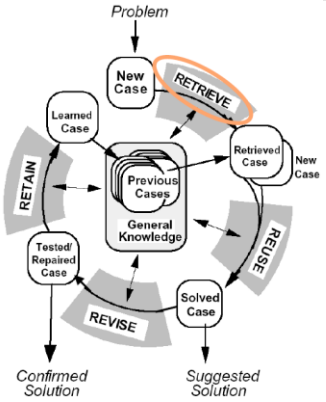
Analoges Schließen: Ähnlichkeiten von mehreren Eigenschaften erlaubt Schluss auf Ähnlichkeit von weiteren Eigenschaften.



Analoger Schluss: Elektron kreist um Kern.

Ein Fall in CBR ist die Beschreibung einer aufgetretenen Problemsituation zusammen mit Erfahrungen, die während der Bearbeitung des Problems gesammelt wurden. Fall enthält Problembeschreibung, Lösung(sversuch), Ergebnis.

11.3.2 CBR Zyklus



- *Retrieve*: Finde ähnliche Fälle.
  - Ähnlichkeitsmaß: Euklidische Distanz, Syntaktische (knowledge-poor) oder semantische (knowledge-intensive) Ähnlichkeiten.
  - Fallbasis Organisation: Array, Baum, Graphen, Indexstruktur, DB
  - Indexierung: Nehme verschiedene Attribute eines Falls als Indizes.
    - Auswahl der Indizes:
      - Manuell
      - Check-Liste: Heuristik zur Erstellung der Indizes
      - Differenzbasiert (lösche sehr allgemeine Attr.)
      - Kombinationen Checkliste und differenzbasiertes Pruning
- *Reuse*: Lösungsadaption.
  - Einfaches Kopieren ohne Adaption, manuell, regelbasiertes Schließen, modellbasiertes Schließen, Planer
- *Revise*: Überprüfung und Verbesserung der Lösung.
  - Lösungsevaluation: Überprüfung durch Simulation oder in realer Welt.
  - Verbesserung/Reparierung
- *Retain*: Gemachte Erfahrung speichern.

11.3.3 Bewertung

- Vorteile
  - Konzeptionell einfach, aber potentiell komplexe Entscheidungsgrenzen
  - Wenig Informationen notwendig
  - Analog zum menschlichen Problemlösen
  - „One-shot“ Learning, einfaches Lernen
- Nachteile
  - Gedächtniskosten
  - Potentiell lange Klassifikation
  - Von Repräsentation abhängig, problematisch bei komplexen Repräsentationen oder bei irrelevanten Eigenschaften

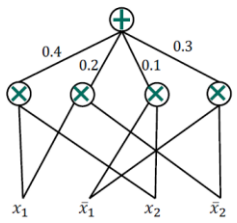
Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	*symbolisch* vs subsymbolisch
Lernvorgang	*überwacht* vs unüberwacht
Beispielgebung	*inkrementell* vs nicht inkr.
Umfang der Bsps	umfachreich vs *gering*
Hintergrundwissen	*empirisch* vs axiomatisch

12 Sum-Product Networks

12.1 SPN-Modell

- Struktur
  - Ein SPN ist ein gerichteter azyklischer Graph mit univariaten Wskverteilungen als Blätter, Summen/Produkten als interne Knoten und gewichteten Kanten.
- Semantik
  - Summenknoten stellen Wahlmöglichkeit der Klasse dar (Mixtur), zB Katze/Hund/Kuh
  - Produktknoten stellen Faktorisierung in Teilkomponenten dar, zB Kopf/Körper/Beine
- Netzwerkpolynom (NP)
  - Indikatorfunktion: $x_i = [X_i] = 1 \Leftrightarrow X_i = true$, 0 sonst. $\bar{x}_i = [\bar{X}_i] = 1 \Leftrightarrow X_i = false$, 0 sonst. Indikatoren stellen Blätter im SPN dar.
  - $NP: f(x) = \sum_x \Phi(\mathrm{x})\Pi(\mathrm{x})$ für Verteilung $\Phi(x)$ von binären boolschen Variablen X und $\Pi(x)$ als Produkt von Indikatoren, die eins sind.
    - Beispiel siehe F12.8
  - Netzwerkpolynom ist wird als Formel für ein SPN Graph umgesetzt.

X_1	X_2	$\Phi(\boldsymbol{X})$
1	1	0.4
1	0	0.2
0	1	0.1
0	0	0.3



$$f(x_1, \bar{x}_1, x_2, \bar{x}_2) = 0.4x_1x_2 + 0.2x_1\bar{x}_2 + 0.1\bar{x}_1x_2 + 0.3\bar{x}_1\bar{x}_2$$

Gute Faktorisierung der Formel ermöglicht tieferes SPN.

- Validität
  - Scope eines Knotens: Menge von Variablen, die im Teilbaum des Knotens als Blätter verwurzelt sind.
  - SPN ist vollständig: Die Kinder von Summenknoten dieselben Scopes haben.
  - SPN ist konsistent: Die Kinder von Produktknoten haben disjunkte Scopes.
  - SON ist valide: Es repräsentiert eine Verteilung. Theorem: valide, wenn vollständig und konsistent.

12.2 Inferenz

Wsk einer Zufallsvariablenbelegung kann durch seetzen der Indikatoren und darauf folgende Vorwärtspropagierung (von unten nach oben) berechnet werden. Siehe F12.17.

- Randwsks: Für X_1, X_2 ist $\mathbb{P}(X_1 = 1)$ gegeben für Indikatorbelegung $x_1 = 1, \overline{x_2} = 0, x_2 = 1, \overline{x_2} = 1$.
- Most Probable Explanation (MPE): Gegeben $X_1 = 1$, welcher Wert ist am wahrscheinlichsten für X_2?
  - Ersetze Summenknoten durch Max-Knoten.
  - Rückwärtspropagierung (oben nach unten): An max-Knoten wird Kind mit höchstem Wert ausgewertet, an Produktknoten alle Kinder. Wahrscheinlichste Belegung ist das Blatt in dem man endet.
- Sampling: Top-Down, an Summenknoten wähle nur ein Kind mit der Kantenwahrscheinlichkeit.
- Diskrete und (kontinuierliche) Variablen: Für jeden Wert V_j ein Indikator $x_i^j = 1 \Leftrightarrow X_i = V_j$, 0 sonst.

12.3 Parameterlernen

- Ansatz: Valides SPN initialisieren, Backpropagation basierte Ansätze mit Gradientenabstieg.
- Generatives Lernproblem: Gegeben SPN mit Scope X, Menge D von Instanzen der Variablen X. Gesucht sind Gewichte, die $\mathbb{P}(x)$ maximieren.
  - Gradientenabstieg: $\nabla \log \mathbb{P}(x) = \nabla \sum_{\mathrm{h}} \mathbb{P}(\mathrm{h}, \mathrm{x})$
  - h = versteckte nicht beobachtbare Variablen
- Diskriminatives Lernen: Gegeben SPN mit Scope $X \cup Y$, Menge D von Instanzen der Variablen X mit Labeln Y. Gesucht sind Gewichte, die $\mathbb{P}(y|x)$ maximieren.
  - Gradientenabstieg: $\nabla \mathbb{P}(y|x) = \nabla \log \frac{\mathbb{P}(y,x)}{\mathbb{P}(x)} = \nabla \log \sum_h \mathbb{P}(y,h,x) - \nabla \log \sum_{y',h} \mathbb{P}(y',h,x)$

- Linker Teil der Subtraktion ist Inferenzansatz auf gelabelten Daten, rechter ist Marginalisierungsansatz für X.

- Backpropagation siehe F12.29ff
- Problem Backpropagation: Durch Produkte (<1) kann bei vielen Schichten der Gradient sehr klein werden, daher: Vanishing Gradient.
  - Harter Gradientenabstieg: Statt alle Teilpfade zu durchlaufen und anteilig aufzuaddieren, nur größten Summand bei +-Knoten nehmen und gesamte Summe damit approximieren.

12.4 Strukturlernen

Erzeuge valides SPN mit Lerndaten. Summen entsprechen Mixturen, jede Mixtur-Komponente ist ein Cluster von Daten. Produkte kombinieren unabhängige Features.

Algorithmus: Clustere ähnliche Lernbeispiele mit Summenknoten, prüfe statistische Unabhängigkeit innerhalb Clustern mit Produktknoten.

Algorithmus ist greedy, top-down und rekursiv. Beispiel auf F12.38ff.

12.5 Anwendungen

zB Datenvervollständigung bei Fotos, Aktionserkennung in Bildern (Örtlicher Zusammenhang wichtig, SPNs kodieren Ortsrelationen).

13 Unüberwachtes Lernen

Ausnutzen von Ähnlichkeiten, um Klassen aus Datenmerkmalen zu schließen.

13.1 k-Means Clustering

Gegeben X Trainingsbeispiele und k als Anzahl der gesuchten Cluster, gesucht Clustering mit minimaler Summe der Entfernungen von Datenpunkten zu Clusterzentren.

Algorithmus: platziere k Punkte als Zentrn, dann iterativ jeweils alle Datenpunkte klassifizieren (nähestes Zentrum) und dann Zentren neu berechnen.

Bewertung: Resultate von initialer Zentrenwahl abhängig, Curse of Dimensionality, korrekte Wahl von k vorausgesetzt.

Fuzzy-k-Means: Jeder Datenpunkt hat eine unscharfe Clusterzugehörigkeit $\in [0,1]$.

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	symbolisch vs *subsymbolisch*
Lernvorgang	überwacht vs *unüberwacht*
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

13.2 Hierarchisches Clustering

Cluster können Subcluster haben. Vgl Vorstellung eines Dendogramms.

- Algorithmus: Agglomerative Hierarchical Clustering
  - Initialisiere Datenpunkte als Cluster, merge nähesten Cluster solange bis Anzahl Cluster erreicht sind oder alle Cluster weiter entfernt sind als Grenzwert.

Einfach, besser als k-Means bei unbekanntem k, erfordert Abstandsfunktion und Grenzwert/Minimale Clusterzahl bekannt. Unabhängig von Initialwerten, aber rauschanfällig.

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	symbolisch vs *subsymbolisch*
Lernvorgang	überwacht vs *unüberwacht*
Beispielgebung	inkrementell vs *nicht inkr.*
Umfang der Bsps	*umfachreich* vs gering
Hintergrundwissen	*empirisch* vs axiomatisch

13.3 Begriffliche Clusters & COPWEB

COBWEB: Lernen durch inkrementelles Aufbauen eines Strukturbaums/Kategoriebaums. Jede Verzweigung steht für Aufteilung in Kategorien. Blätter sind speziellsten Kategorien.

- *Maß für Clusternützlichkeit*: Cluster hat eine hohe Nützlichkeit, wenn für alle Beispiele darin die (unbekannten) Attributwerte mit hoher Wsk vorhersagbar sind, und für alle Beispiele mit gegebenen Attributwerten die Beispiele mit hoher Wsk die Zugehörigkeit vom Beispiel zum Cluster bestimmt werden kann. (Cluster Utility)

Beispiel und Algorithmus siehe F13.32.

- Vorgehensweise:
  - Beispiel in Knoten einfügen: Wenn Knoten Blatt ist, Blatt durch neuen Knoten ersetzen und daran neues Beispiel und altes Blatt einfügen. Wenn Knoten Nachfolger hat, nur Parameter auf Knoten anpassen.
  - Wenn Bsp in keine Klasse passt, neuen Knoten erzeugen.
  - Konzept verallgemeinern: Zwei Knoten vereinen.
  - Konzept war zu allgemein?: Knoten aufteilen.

Typ der Inferenz	*induktiv* vs deduktiv
Ebenen des Lernens	*symbolisch* vs subsymbolisch
Lernvorgang	überwacht vs *unüberwacht*
Beispielgebung	*inkrementell* vs nicht inkr.
Umfang der Bsps	umfachreich vs *gering*
Hintergrundwissen	*empirisch* vs axiomatisch

14 Evolutionäre Algorithmen

- Grundalgorithmus: Solange nicht optimal, selektiere Eltern, generiere Nachkommen, bewerte Fitness und selektiere überbleibende Population.
- *Generieren von Nachkommen*: Exploration (Erforschung des Hypothesenraums) oder Exploitation (lokale Optimierung)
- *Mutation*: Geene von Individuen werden speziell kodiert, in der Kodierung werden einzelne Elemente abgewandelt. zB Bitinversion oder Sequenz Inversion
- *Rekombination*: Eigenschaften mehrerer Eltern werden gemischt.
  - Diskrete Rekombination: xxxxxx+yyyyyy=xyxxyyxyyxyx
  - Intermediäre Rekombination: $x + y = z$, $z_i = \frac{x_i + y_i}{2}$
  - Crossover: Zwei Eltern erzeugen zwei nachkommen
    - Single-point Crossover: xxxx+yyyy=xxyy, yyxx
    - Two-point crossover: xxxx+yyyy=xyyx, yxxy
    - Uniform crossover: xxxx+yyyy=xyxy, yxyx
- *Selektion*: Selektion der Eltern für Mating, und Selektion der Restpopulation
  - Probleme: Genetischer Drift (manche Individueen vermehren sich zufällig mehr als andere), Crowding/Ausreißerproblem (fitte Individuen dominieren Population)
  - Mating
    - Inselmodell (lokal): Nur manchmal werden Individuen ausgetauscht, weitestgehend getrennte Evol.
    - Nachbarschaftsmodell: Nachkommen dürfen nur von Eltern erzeugt werden, die in ihrer Nachbarschaft am besten sind
    - Einfache Menge (global): Global besten entwickeln sich, andere werden unterdrückt
  - Populationsmitglieder
    - Populationsgröße: Konstant? Wie viele Nachkommen? Wie viele genutzte Eltern? Welche?
    - Daumenregel: Das 1/4 der Population sollte ¾ der Nachkommen erzeugen.
  - Restpopulations-Selektion: mit welcher Wsk wird x reselektiert?
    - Fitnessbasierte Selektion: $\mathbb{P}(x) = \frac{f(x)}{\Sigma_{x' \in Pop} f(x')}$
    - Rang basierte Selektion: Wie fitnessbasiert, aber statt f wird abhängig vom Rang von x selektiert. Rang kann durch weitere Funktion angepasst werden (zB „die besten k“ oder mit Exponentialfkt)
    - Turnier Selektion: Wähle für jedes zu erzeugende Individuum bestehende Individuen, belohne das beste. Wähle dann die mit höchster Belohnung.
- Genetische Programmierung
  - Erzeuge optimierte Programme, Programm wird als Baum dargestellt. Rekombination: Austausch von Teilbäumen.
- Diskussion
  - Gute Parallelisierbarkeit
  - Sehr rechenintensive Ansätze
  - Vorwissen kann durch Initialpopulation integriert werden

Übersicht