Thema: Algorithmische Lerntheorie

HS Algorithmen SS 2004
Dozent: Prof. H. Alt
Referenten: Sebastian Frielitz, Anna Kress

Einleitung

In den späten 50er Jahren gab es die ersten Versuche, Programme zu entwickeln, die Arbeiten verrichten sollten, von denen man annimmt, dass sie Intelligenz erfordern. Das ergeizige Ziel einiger Forscher war es, letztlich Computer zu bauen, die die Denkfähigkeit eines Menschen besitzen (oder sogar übertreffen) sollten.
Ein wichtiges Kennzeichen von Intelligenz ist die *Lernfähigkeit*. Entsprechend wurde versucht, Programme zu entwickeln, die ihr Verhalten abhängig von Erfahrung verändern konnten. In anderen Worten, diese Programme sollten *lernfähig* sein.
Der Vorteil lernfähiger Programme zeigt sich, wenn man sich Probleme vor Augen führt, die nur schwer (wenn überhaupt) algorithmisch beschreibbar sind. Ein Beispiel wäre das Erlernen einer Sprache, also die Fähigkeit, einen korrekten Satz in dieser Sprache zu verstehen und zu bilden. Dieses würde im Fall der lernfähigen Programme nicht über eine harte Kodierung der semantischen und syntaktischen Regeln zur Bildung eines korrekten Satzes erfolgen, sondern über den Kontakt mit vielen Beispielen (und eventuell mithilfe von einigen vorgegebenen "Daumenregeln").
Die *Algorithmische Lerntheorie* (Computational Learning Theory) ist der Zweig der theoretischen Informatik, welcher formal studiert, wie man Computerprogramme entwickeln kann, die lernfähig sind. Dazu stellt die Lerntheorie einen formalen Rahmen bereit, in welchem man präzise Fragen zur Performance der unterschiedlichen Algorithmen stellen kann. Damit kann man sie auf ihre "Lerngüte" und ihre Berechnungseffizienz vergleichen.
Drei Schlüsselaspekte müssen dabei formalisiert werden:

1) Die Art und Weise, wie der Lernalgorithmus mit der Umwelt interagiert.
2) Die Definition, wann eine Lernaufgabe erfolgreich war.
3) Die Definition der Effizienz eines Lernalgorithmus, wobei zwei Bereiche unterschieden werden müssen: die Effizienz der Datennutzung (*sample complexity*) und die Laufzeit zur Verarbeitung der Beispiele (*time complexity*).

1. Konzeptlernen

Die grundlegende Idee hinter den in den nächsten Kapiteln vorgestellten Lernmodellen ist das Konzeptlernen. Konzeptlernen ist dabei der Prozess, in dem der Lernende versucht, eine gute Näherung (Hypothese) an ein unbekanntes Zielkonzept (target concept) zu konstruieren. Dabei erhält er eine Anzahl von „etikettierten" Beispielen (labeled examples) und eventuell auch einige Vorabinformationen (z.B. „Daumenregeln") über das Zielkonzept. Hier besteht eine gewisse Analogie zum Interpolationsverfahren in der Mathematik, bei dem einige Punkte einer Funktion bekannt sind, und die restlichen „erraten" werden sollen. Die Vorabinformation könnte hier darin bestehen, dass man weiss, dass die gesuchte Funktion ein Polynom 5-ten Grades ist.

Ein Zielkonzept wäre z.B. das Konzept "Verb in der deutschen Sprache". Dieses würde eine Menge aus allen Verben der deutschen Sprache sein, d.h. Zielkonzept = {"gehen", "lernen", …}.

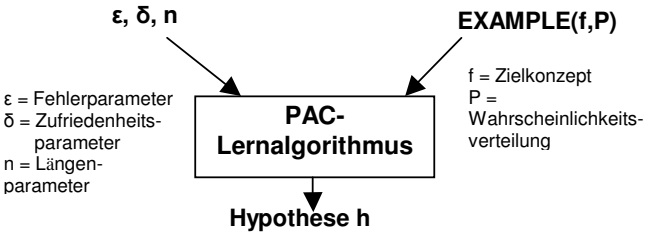
1.1. Notation von Konzeptlernen:

- *Domäne X* (instance space bzw. domain) – eine Menge, die alle möglichen Objekte enthält (in unserem Beispiel alle Wörter der deutschen Sprache)
- *Konzept f* (concept) – 2 Sichtweisen:
  → mengentheoretisch: f ist Teilmenge von Domäne X,
  (Konzept Substantiv, Konzept Verb, etc.)
  → funktional: f(x), x ϵ X, f ist boolsche Funktion:
  f gibt an, ob x dem Zielkonzept entspricht oder nicht
- *positives/negatives Beispiel x* für ein Konzept f – x ϵ X
  zusammen mit einem „Etikett" (label): positiv: f(x) = 1, negativ: f(x) = 0
- *Konzeptklasse C* (concept class)– Menge der Konzepte, typischerweise (aber nicht immer) kennt der Lernende C

2. PAC-Lernmodell (Probably Approximately Correct)
Das PAC-Lernmodell würde 1984 von Valiant vorgeschlagen, womit er auch das Feld der Algorithmischen Lerntheorie begründete.
Im PAC-Modell werden *die Beispiele, die der Lernalgorithmus erhält, zufällig gemäß einer unbekannten Wahrscheinlichkeitsverteilung P erzeugt*. (Durch die zufällige Wahl der Beispiele wird das Problem gemildert, dass die Lernsituation eine andere als die Anwendungssituation ist.)
Ausserdem wird *ein Maß für die Güte der produzierten Hypothese* eingeführt.
Man unterscheidet im PAC-Lernmodell (im Gegensatz zum Online-Lernmodell) zwischen einer *Lern-* und einer *Anwendungsphase*.

2.1 Formale Definition des PAC-Lernmodells
A ist ein PAC-Lernalgorithmus für eine Konzeptklasse C, wenn

- A als Eingabe $\varepsilon \in (0,1]$, $\delta \in (0,1]$ und $n \in N$ erhält
- A eine Routine *EXAMPLE(f, P)* aufrufen kann, die "etikettierte" Beispiele für Konzepte $f \in C$ zurückliefert. Die Beispiele werden zufällig entspr. einer willkürlichen und unbekannten Wahrscheinlichkeitsverteilung P über der Domäne X ausgewählt.
- Für alle Konzepte $f \in C$ und für alle Wahrscheinlichkeitsverteilungen P gibt A mit Wahrscheinlichkeit von mind. $(1- \delta)$ ein Konzept h aus, so dass h höchstens den Fehler ε hat.



Fehlerparameter ε: spezifiziert den Fehler, der in einer guten Annäherung erlaubt ist.
Zufriedenheitsparameter δ: bestimmt die Wahrscheinlichkeit für das Konstruieren einer guten Näherung. (Wenn z.B. der Algorithmus nur negative Beispiele präsentiert bekommt, kann er keine gute Hypothese konstruieren.)
Beispiel: $\varepsilon = 0{,}05$ und $\delta = 0{,}01 \rightarrow$ Der Algorithmus gibt mit einer Wahrscheinlichkeit von mind. 99% eine Hypothese aus, die mind. 95% der Beispiele genauso klassifiziert wie das Zielkonzept.
Längenparameter n: schränkt die Portion der Domäne X ein, die für den Lernenden relevant ist. (Z.B. könnte man für das Zielkonzept "Verb" einschränken, dass die Länge des Verbs höchstens 10 beträgt.)

Ein PAC-Lernalgorithmus ist *effizient*, wenn
1. die Anzahl der zum Lernen benötigten Beispiele (= *sample complexity*) und
2. die Zeit zur Verarbeitung dieser Beispiele (= *time complexity*)
polynomiell in n, 1/ε und 1/δ ist.
Zu beachten: 1. und 2. sind voneinander unabhängig, d.h. ein Lernalgorithmus kann auch nur effizient bezüglich der samle complexity oder nur effizient bezüglich der time complexity sein. (Dann ist es kein PAC-Lernalgorithmus.)

Bei der *time complexity* fallen dabei ins Gewicht: die Zeit, die erforderlich ist, um ein etikettiertes Beispiel anzufordern, die Zeit, die erforderlich ist, um das Beispiel einzulesen und zu bearbeiten (hängt mit der Repräsentationsgröße des Beispiels zusammen) und die Zeit, die erforderlich ist, um die Hypothese zu bilden (hängt mit der Repräsentationsgröße der Hypothese zusammen).
Sample complexity: Man kann sie sich als eine Funktion $s(\varepsilon, \delta, n)$ vorstellen, wobei s die maximale Anzahl von Aufrufen von EXAMPLE(f, P) durch den Lernalgorithmus A ist. Das Maximum wird über alle Läufe von A auf allen möglichen Eingaben ε, δ, n über alle Konzepte aus Konzeptklasse C, alle möglichen Sequenzen, die EXAMPLE ausgibt und alle Wahrscheinlichkeitsverteilungen bestimmt. Wenn kein endl. Maximum existiert: $s = \infty$.

2.2 Fazit:
Die *Vorteile*, die dem PAC-Modell zum Durchbruch verholfen haben, sind: die Analysierbarkeit von Lernalgorithmen, die man durch das Modell in die Hand bekommt und die Tatsache, dass es sich dabei um ein vernünftiges Modell des natürlichen Lernprozesses handelt.
Die *Nachteile* bestehen darin, dass der Algorithmus vor seinem praktischen Einsatz eine Lernphase durchmachen muss; dass die Anforderungen an einen PAC-Lernalgorithmus höher sind, als es die Praxis vielleicht erfordert (er muss z.B. über alle ε, δ und P funktionieren) und die nicht vorhandene Robustheit gegen Rauschen (noise) bei der Etikettierung der Beispiele (d.h. Beispiele werden falsch etikettiert).

2.3 VC-Dimension (nach Vapnik und Chervonenkis 1971)
Sei C eine Konzeptklasse über eine Domäne X.
Wir sagen, C *zerschmettert* (shatters) eine endl. Menge $S \subseteq X$, wenn für jede der $2^{|S|}$ Teilmengen $S' \subseteq S$ es ein Konzept $f \in C$ gibt, welches alle Elemente von S' und keine von S - S' enthält.
Mit anderen Worten: Für jedes der $2^{|S|}$ möglichen "Zuordnungen" von S gibt es ein $f \in C$, welches dieses „Zuordnungen" realisiert.
Die VC-Dimension von C = *VCdim(C)* ist das kleinste d, für welches keine Menge von d + 1 Beispielen existiert, welche von C zerschmettert wird.

3. Exakte und Fehlerbegrenzte Lernmodelle (Exact and Mistake Bounded Learning Models)
Das PAC Lernmodell ist ein "hintereinander ausführendes" Modell (batch model). Es besteht aus einer vorangestellten Trainingsphase (training phase) und einer danach laufenden Ausführungsphase (performance phase). In den nun vorgestellten Modellen gibt es diese Trainingsphase nicht mehr, sondern es muss während der Ausführungsphase gelernt werden.

3.1 Das On-Line Lernmodell für den allgemeinen Fall
Ein Onlinealgorithmus besteht aus einer *Lernsession* (learning session), die aus einem Set von *Versuchen* (trials) besteht.
Jeder *Versuch* besteht aus:

- Lerner bekommt ein unbenanntes Element $x \in X$
- Lerner wendet die momentane Hypothese an um den Wert p(x) für das Zielkonzept $f \in C$ vorherzusagen.
- Der Lerner erfährt den richtigen Wert f(x)

Die Qualität eines Lerners wird durch Fehlvorhersage-Funktionen angegeben:

- Quadratisch $l_2(p(x), f(x)) = (f(x) - p(x))^2$ (square loss)
- Logarithmisch $l_{\log}(p(x), f(x)) = -f(x)\log p(x) - (1 - f(x))\log(1 - p(x))$ (log loss)
- Absolut $l_1(p(x), f(x)) = | f(x) - p(x) |$ (absolute loss)

Für die hier betrachteten Fälle, das ein Lerner nur eine boolsche Ausgabe hat, wird die Absolute Fehlvorhersage-Funktion verwendet.
Das Ziel des Lerners ist: so wenig Fehlvorhersage wie möglich gerechnet über alle Vorhersagen zu machen.

3.1.1 Halving Algorithmus Weighted Majority Algorithmus
Halving Algorithmus:

- Alle Hypothesen aus *C* als Kandidaten *CAND*
- In jedem Schritt werden alle Kandidaten befragt:
  - $CAND_1$ – Kandidaten die 1 ausgeben
  - $CAND_0$ – Kandidaten die 0 ausgeben
- Wenn $CAND_1 > CAND_0$ dann h(x)=1 sonst h(x)=0
- Wenn h(x) nicht gleich f(x) dann CAND=CAND\{max(*CAND1,CAND0*)

Man kann leicht sehen, das bei jeder Falschvorhersage die Anzahl der Kanditaten mindestens halbiert wird und der Algorithmus so eine Qualität von lg(n) hat.

Weighted Majority Algorithmus

Weighted Majority Algorithmus (WM)
Sei w_i zugehöriges Gewicht zu A_i
Für 1 <= i <= n
Sei $w_i = 1$
Vorhersage für Instanz $x \in X$:
Seien A_0 die Experten die 0 vorhersagen
Seien A_1 die Experten die 1 vorhersagen
Sei $q_0 = \sum_{A_i \in A_0} w_i$
Sei $q_1 = \sum_{A_i \in A_1} w_i$
Vorhersage 0 wenn $q_0 \geq q_1$
Wenn ein Fehler gemacht wurde:
Sei y die Vorhersage
Für 1 <= i <= n
Wenn A_i's Vorhersage für x nicht y ist
Sei $w_i = w_i \cdot \beta$
Informiere A_i

Der *Algorithmus A* hat einen Pool von *„Experten" A_i*. Einer dieser Experten liefert für das Problem gute Ergebnisse, A weiß nur nicht welcher.
Jedem Experten ist ein *Gewicht* w_i zugeordnet, das die Vertrauenswürdigkeit des Experten angibt.
Wenn ein Experte ein falsches Ergebnis vorhersagt, dann wird sein Gewicht um den *Faktor* β ($0 \leq \beta < 1$) verringert. Um gegen Rauschen sicher zu sein muss, $\beta > 0$ sein.

4. Query Learning Model

Wird als *exaktes Lernmodell* bezeichnet. Das Ziel ist hierbei, durch Fragen exakt zu lernen, wie eine boolsche *Zielfunktion f* funktioniert.
Die *Zielfunktion* wird von einer bekannten *Konzeptklasse C* genommen.
Der Lerner hat zwei Möglichkeiten zu fragen, um die *Zielfunktion* zu ermitteln.

1. Die *Mitgliedschaftsnachfrage* (membership query): Wobei der Lerner eine Instanz x ϵ X wählt und dafür den Wert f(x) erfährt.
2. Die *Vergleichsnachfrage* (equivalence query): Wobei der Lerner eine *Kanditatenfunktion h* präsentiert und erfährt, ob es die richtige ist $h \equiv f$ (wenn ja, dann ist der Lerner fertig), ansonsten bekommt er ein *Gegenbeispiel x*, für das gilt g(x)≠f(x).

Query Learning Beispiel für monotone DNF

Ein *Primimplikant* t ist eine erfüllende Kombination von Literalen für eine *Formel f*, so das kein Unterterm von t f erfüllt.
Beispiel: $f = (a \wedge c) \vee (b \wedge \bar{c})$ hat als *Primimplikant* $a \wedge c, b \wedge \bar{c}, a \wedge b$.
Die Anzahl von Primimplikanten einer generellen DNF kann exponentiell größer sein als die Anzahl an Termen. Bei der *monotonen DNF* hingegen ist die Anzahl an Primimplikanten nicht größer als die Anzahl der Terme.
$f = (a \wedge c) \vee (b \wedge c)$ hier z.B. gibt es nur zwei Primimplikanten, und das sind genau die beiden Terme.

Angluins Algorithmus

h <- false
Endlosschleife:
Vergleichsnachfrage für h
Wenn „ja" dann gibt h aus und fertig
Ansonsten Sei $x=b_1,b_2,...,b_i$ ein Gegenbeispiel
Sei $t = \wedge_{b_i=1} y_i$
Für i = 1,...,n
Wenn b_i=1 Mitgliedschaftsnachfrage für x mit i-tem bit invertiert
Wenn „ja" t=t \ {y_i} und $x=b_1...b_i...b_n$
Sei $h = h \vee t$

Ein Query Learning Algorithmus ist der *Algorithmus von Angluin*, mit dem monotone DNF gelernt werden kann. Die Idee dahinter ist, dass man über alle Belegungen der boolschen Variabeln iterieren kann und mit jeder Iteration einen *Primimplikanten* für *f* erhält. Dabei läuft die Schleife genau einmal über jeden *Primimplikanten* von *f*.

Literatur: M. J. Atallah, Ed.: Algorithms and Theory of Computation Handbook, CRC Press LLC, 1999; Natarajan, Balas K.: Machine learning: a theoretical approach, San Mateo, 1991; www.cis.temple.edu/~ingargio/cis587/readings/pac.html, 17.06.04