ALGORITHMISCHES LERNEN

SS 94

Georg Schnitger
Christine Schmetzer

Universität-Gesamthochschule Paderborn
Fachbereich 17
33095 Paderborn

Inhaltsverzeichnis

1 Einführung

Wir beschäftigen uns mit dem Problem des **Konzeptlernens aus Beispielen und Gegenbeispielen**. Dabei unterscheiden wir zwei Lernmodelle:

- Passives Lernen: Die Konzeptklasse ist dem Lerner bekannt. Der Lerner fordert Beispiele (auch positive Beispiele genannt) und Gegenbeispiele (auch negative Beispiele genannt) an, um ein unbekanntes Konzept der Konzeptklasse zu erlernen. Im PAC-Modell werden die Beispiele gemäß einer unbekannten Verteilung D generiert. Ziel des Lerners ist es, eine Hypothese aufzustellen, die für die gegebene Verteilung *wahrscheinlich approximativ richtig* (probably approximateley correct) ist. Ein unbekanntes Konzept der Konzeptklasse soll also an Hand der klassifizierten Beispiele zumindest approximativ erlernt werden.

  Wir betrachten auch das verteilungs-spezifische Lernmodell, in dem die Verteilung D dem Lerner bekannt ist.

- Aktives Lernen: Auch hier ist die Konzeptklasse dem Lerner bekannt. Der Lerner hat zusätzlich die Möglichkeit, Fragen zu stellen. Dies können Fragen der folgenden Form sein:

  - Elementfragen: Ist die Eingabe x ein positives Beispiel?
  - Äquivalenzfragen: Ist die jetzige Hypothese korrekt? Wenn nein, erwartet der Lerner ein Gegenbeispiel.

1.1 Überblick

An dieser Stelle geben wir einen Überblick über die folgenden Kapitel der Vorlesung.

Im passivem Lernmodell versucht der Lernalgorithmus, eine Hypothese auszugeben, die mit den angeforderten klassifizierten Beispielen konsistent ist. In Kapitel 2 untersuchen wir, wieviele Beispiele angefordert werden müssen, damit eine konsistente Hypothese wahrscheinlich approximativ richtig ist. Wir erhalten eine recht befriedigende Antwort mit Hilfe der Vapnik-Chervonenkis Dimension der Konzeptklasse.

In Kapitel 2 kümmern wir uns noch nicht darum, wie eine konsistente Hypothese effizient gefunden werden kann. Diese Frage greifen wir in Kapitel 3 auf: In Kapitel 3.1 entwickeln wir effiziente Lernalgorithmen für eine Reihe wichtiger Konzeptklassen.
Wir lernen aber auch ein erstes Beispiel für eine Konzeptklasse kennen, deren Konsistenzproblem („finde ein Konzept der Konzeptklassse, das mit den vorgegebenen Beispielen konsistent ist“) sehr wahrscheinlich nicht effizient lösbar ist: Wir zeigen, daß das Existenzproblem

> „Gibt es ein konsistentes Konzept zu einer vorgegebenen Menge klassifizierter Eingaben?“

NP-vollständig ist. Das bedeutet: **Wenn** die Konzepte der vorgegebenen Konzeptklasse als Hypothesen auszugeben sind, gibt es mit hoher Wahrscheinlichkeit keinen effizienten Lernalgorithmus. Wir haben ein repräsentations-abhängiges negatives Ergebnis erhalten, da es nicht ausgeschlossen ist, daß konsistente Hypothesen aus einer **anderen** Konzeptklasse effizient gefunden werden können.

In Kapitel 3.2 betrachten wir ein Paradigma für den Entwurf von Lernalgorithmen, nämlich Occam-Algorithmen: Wenn ein Lernalgorithmus **relativ viele** Beispiele mit einer **relativ kurzen, konsistenten** Hypothese erklärt, wird diese Hypothese wahrscheinlich approximativ richtig sein.

In Kapitel 4 erzielen wir repräsentations-unabhängige negative Ergebnisse: Wir lernen Konzeptklassen kennen, die in aller Wahrscheinlichkeit nicht effizient lernbar sind, egal wie die Hypothesenklasse gewählt wird. Diese Ergebnisse können wir nachweisen, indem wir das Lernproblem mit dem Brechen kryptographischer Codes in Verbindung bringen.

In Kapitel 8 betrachten wir eine Variante von PAC-Algorithmen: Lernalgorithmen, deren Hypothesen nur eine sehr schwache Voraussagekraft besitzen. Wir zeigen, daß jeder effiziente, schwache PAC-Algorithmus in einen effizienten, starken PAC-Algorithmus umgewandelt werden kann. Der benutzte „Boosting-Algorithmus“ ist dabei von allgemeinem Interesse. Dieses Ergebnis zeigt, daß die Komplexitätklasse aller im PAC-Sinne effizient-lernbaren Konzeptklassen **nicht** von der Definition des Approximationsfehlers abhängt.

Das Ziel einer Theorie des algorithmischen Lernens ist somit eine Katalogisierung in effizient lernbare und nicht-effizient lernbare Probleme. In Kapitel 5 diskutieren wir die vielleicht bekannteste Konzeptklasse, für die es eine solche Katalogisierung nicht gibt: Die Konzeptklasse der aussagenlogischen Formeln in disjunktiver Normalform (DNF-Formeln). Wir besprechen einen „fast-effizienten“ Lernalgorithmus, der DNF-Formeln approximativ erlernt, wenn die Beispiele gemäß der Gleichverteilung generiert werden.

Deterministische endliche Automaten sind im passiven Lernmodell sehr wahrscheinlich nicht effizient lernbar. Deshalb beschäftigen wir uns in Kapitel 6 mit aktivem Lernen. In Kapitel 6.1 entwickeln wir einen effizienten Lernalgorithmus für endliche Automaten, wenn uns Element- und Äquivalenzfragen zur Verfügung stehen. Wenn wir beim Erlernen der Automaten eine Elementfrage stellen, bedeutet dies, daß eine Rückkehr zum Anfangszustand des Automaten (ein Reset) problemlos möglich ist. Wir betrachten deshalb in Kapitel 6.2 auch das Lernproblem ohne Reset.

Neben der Konzeptklasse der DNF-Formeln ist auch die Konzeptklasse der Entscheidungsbäume ein wichtiges Beispiel für eine Konzeptklasse, deren Komplexität bis heute nicht bekannt ist. In Kapitel 7 entwickeln wir zuerst einen „fast-effizienten“ Lernalgorithmus für Entscheidungsbäume im passiven Lernmodell. In Kapitel 7.2 benutzen wir dann die Fourier-Transformation, um Entscheidungsbäume effizient mit Elementfragen zu erlernen.

1.2 Literatur

M. Anthony und N. Biggs: *Computational learning theory*, **Cambridge Tracts in Theoretical Computer Science 30, 1992**

M. J. Kearns: *The computational complexity of machine learning*, **MIT Press, 1990**

B.K. Natarajan: *Machine learning, a theoretical approach*, **Morgan Kaufmann, 1991**

R. Schapire: *The design and analysis of learning algorithms*, **MIT Press, 1992**

1.3 Notationen

Zunächst führen wir einige grundlegende Definitionen ein.

- Für eine Menge X bezeichnen wir mit

  $$\mathcal{P}(X) = \{Y \mid Y \subseteq X\}$$

  die Potenzmenge von X.

- Für ein endliches Alphabet Σ bezeichnet Σ^n die Menge aller Worte der Länge n über Σ.
  Wie üblich ist $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ und $\Sigma^+ = \bigcup_{n=1}^{\infty} \Sigma^n$.
  Die Menge aller Worte über Σ der Länge höchstens n ist $\Sigma^{\leq n} = \bigcup_{k=0}^{n} \Sigma^k$.

- Die natürlichen Zahlen definieren wir als $\mathbb{N} = \{0, 1, \ldots\}$, mit $\mathbb{Z} = \{\ldots, -1, 0, 1 \ldots\}$ bezeichnen wir die Menge der ganzen Zahlen, und $\mathbb{R}$ bezeichnet die Menge der reellen Zahlen.

- Asymptotisches Wachstum:

  Es seien $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Dann sagen wir

  f wächst asymptotisch nicht schneller als g: $f = O(g) \Leftrightarrow f(n) \leq c \cdot g(n)$ für $n \geq n_0$ und eine positive Konstante c.

  f wächst asymptotisch langsamer als g: $f = o(g) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

  f wächst asymptotisch schneller als g: $f = \omega(g) \Leftrightarrow g = o(f)$.

  f wächst asymptotisch mindestens so schnell wie g: $f = \Omega(g) \Leftrightarrow g = O(f)$.

  f und g wachsen gleich schnell: $f = \Theta(g) \Leftrightarrow f = O(g)$ und $g = O(f)$.

- Ein Konzept c über Σ ist eine Teilmenge von Σ^*.

  Eine Konzeptklasse $\mathcal{C}$ über Σ ist eine Menge von Konzepten.

  Jeder Konzeptklasse $\mathcal{C}$ ist eine Namensfunktion

  $$Name : \mathcal{C} \to \mathcal{P}(\{0,1\}^*)$$

  zugewiesen. Dabei fordern wir:

  - Für jedes Konzept $c \in \mathcal{C}$ ist die zugewiesene Namensmenge $Name(c)$ nichtleer, d.h. jedes Konzept erhält mindestens einen (nichtleeren) Namen.
  - Zwei verschiedene Konzepte $c_1, c_2 \in \mathcal{C}$ erhalten keinen gemeinsamen Namen. Mit anderen Worten
  $$Name(c_1) \cap Name(c_2) = \emptyset.$$

- Eine Hypothese h über Σ ist eine Teilmenge von Σ^*. Analog zum Konzeptbegriff werden die Begriffe Hypothesenklasse $\mathcal{H}$ und Namensfunktion $Name$ definiert.

Wir sind an den folgenden Konzeptklassen interessiert:

Beispiele

B1 $Monom_n$ bezeichnet die folgende Konzeptklasse: Für jedes Monom m über den Literalen $x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n$ ist die Menge der Wahrheitsbelegungen

$$wahr(m) = \{y \mid y \in \{0,1\}^n,\ y \text{ erfüllt } m\}$$

ein Konzept. (Die Konzeptklasse $Monom_n$ besteht also aus den Mengen $wahr(m)$, für alle Monome m über den Literalen $x_1, \neg x_1, \ldots, x_n, \neg x_n$.) Die Namensfunktion $Name$ weist dem Konzept $wahr(m)$ eine binäre Kodierung des Monoms m zu. Die leere Menge $wahr(m) = \emptyset$ hat eine Sonderstellung: Alle unerfüllbaren Monome können als Namen dienen; wir einigen uns auf eine Binärkodierung des unerfüllbaren Monoms $x_1 \neg x_1$.
(Ein Monom m ist eine Konjunktion von Literalen; Literale sind Variablen und negierte Variablen.)

B2 Die Konzeptklasse $k\text{-}CNF_n$ besitzt für jede aussagenlogische Formeln in konjunktiver Normalform mit höchstens k Literalen aus $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ pro Klausel ein Konzept, und zwar die „Menge der erfüllenden Wahrheitsbelegungen“. Der Name eines Konzepts $wahr(\alpha)$ ist die Binärkodierung der entsprechenden Formel α:

$$Name(wahr(\alpha)) = \{ \text{ Kodierung}(\beta) \mid \begin{array}{l} \alpha \Leftrightarrow \beta \text{ und } \beta \text{ ist eine } k\text{-}CNF_n \\ \text{über } \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}\}. \end{array}$$

B3 $k\text{-}DNF_n$ ist die entsprechende Konzeptklasse für Formeln in disjunkter Normalform.

B4 Die Konzeptklasse k-Klauseln CNF_n besitzt für jede aussagenlogische Formel in konjunktiver Normalform mit höchstens k Klauseln über den Literalen $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ ein Konzept, nämlich die Menge der erfüllenden Wahrheitsbelegungen. Der Name des Konzepts ist die Binärkodierung der entsprechenden Formel.

B5 k-Term DNF$_n$ ist die entsprechende Konzeptklasse für Formeln in disjunkter Normalform.

B6 Die Konzeptklasse CNF_n besitzt für jede Formel in konjunktiver Normalform über den Literalen $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ die entsprechende Menge der erfüllenden Wahrheitswerte als Konzept.

B7 DNF_n ist die entsprechende Konzeptklasse für Formeln in disjunkter Normalform.

B8 Die Konzeptklasse $Boolean_n$ besteht aus allen Teilmengen von $\{0,1\}^n$, d.h. jede Teilmenge bildet ein Konzept. Es bieten sich verschiedenste Namensfunktionen an: Konjunktive Normalform, disjunktive Normalform oder Schaltkreise, die genau die vorgegebene Teilmenge akzeptieren. (Hierbei wählen wir das Schaltkreismodell mit UND-, ODER- und NICHT-Bausteinen.)

B9 $Threshold_n$ besitzt für jede Thresholdfunktion

$$f(x) = \begin{cases} 1 & \sum_{i=1}^{n} w_i x_i \geq t \\ 0 & \text{sonst}, \end{cases}$$

mit $w_1, \ldots, w_n, t \in \mathbb{R}$, die Menge der erfüllenden Wahrheitsbelegungen

$$wahr(f) = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n) \in \{0,1\}^n,\ f(x_1, \ldots, x_n) = 1\}$$

als Konzept.

B10 k-$Threshold_{n,s}$ besitzt für jeden Schaltkreis der Tiefe k, dessen Bausteine Thresholdfunktionen sind, die entsprechende Menge der erfüllenden Wahrheitsbelegungen als Konzept. Wir nehmen dabei an, daß jeder Schaltkreis höchstens n Eingabebits erhält und höchstens s Bausteine besitzt.

B11 Sym_n besitzt für jede symmetrische Funktion $s : \{0,1\}^n \rightarrow \{0,1\}$ die Menge

$$wahr(s) = \{x \mid x \in \{0,1\}^n,\ s(x) = 1\}$$

als Konzept.
(Eine Funktion $s : \{0,1\}^n \rightarrow \{0,1\}$ heißt symmetrisch, falls für jede Permutation π

$$s(x_1, \ldots, x_n) = s(x_{\pi(1)}, \ldots, x_{\pi(n)})$$

gilt.)

B12 $Decision_{n,s}$ besitzt für jeden Entscheidungsbaum T über den Booleschen Variablen $x_1, \ldots, x_n$ mit höchstens s Knoten die Menge

$$wahr(T) = \{x \mid x \in \{0,1\}^n,\ x \text{ erfüllt } T\}$$

als Konzept.
(Ein Entscheidungsbaum ist ein binärer Baum, dessen Knoten mit einem Element aus $\{x_1, \ldots, x_n\}$ und dessen Blätter mit den Wahrheitswerten 0 oder 1 markiert sind. Eine Eingabe $y = (y_1, \ldots, y_n) \in \{0,1\}^n$ wird an der Wurzel angelegt. Erreicht die Eingabe den Knoten v mit Markierung x_i, wird sie zum linken (rechten) Kind weitergeleitet, falls $y_i = 0$ ($y_i = 1$) ist. Wir sagen, daß y den Baum erfüllt, wenn y ein Blatt erreicht, das mit 1 markiert ist.)

B13 k-$Liste_{n,s}$ besitzt für jede Entscheidungsliste über den Variablen $x_1, \ldots, x_n$ die entsprechende Menge der erfüllenden Wahrheitsbelegungen als Konzept.
(Eine Entscheidungsliste ist durch einen Vektor der Form $((m_1, b_1), \ldots, (m_e, b_e), b_{e+1})$ (mit $e \leq s$) spezifiziert. Jedes m_i, $1 \leq i \leq e$, ist hierbei ein Monom von k Literalen. Die Wahrheitswerte b_i werden aus der Menge $\{0,1\}$ gewählt. Für eine Eingabe $y = (y_1, \ldots, y_n) \in \{0,1\}^n$ wird die Ausgabe b_j errechnet, wobei das kleinste j gewählt wird, so daß y das Monom m_j erfüllt. Wenn y keines der e Monome erfüllt, wird die Ausgabe b_{e+1} gegeben.)

B14 $Aut_{n,\Sigma}$ besitzt für jeden deterministischen endlichen Automaten mit höchstens n Zuständen die entsprechende Menge aller akzeptierten Worte als Konzept. Dabei ist Σ das Eingabealphabet des Automaten.

B15 $Rechteck_{n,d}$ besitzt für jedes achsenparallele Rechteck R in $\mathbb{R}^d$ ein Konzept. Dieses Konzept besteht aus allen Vektoren $x \in \mathbb{Z}^d$ (mit $-2^n \leq x_i \leq 2^n$), die in R enthalten sind.

1.4 Das PAC-Lernmodell

Gegeben sei die Konzeptklasse $\mathcal{C}$ sowie die Hypothesenklasse $\mathcal{H}$. Der Lerner versucht, ein unbekanntes Konzept $c \in \mathcal{C}$ durch eine Hypothese $h \in \mathcal{H}$ zu erklären. Dazu steht der folgende Typ von Lernalgorithmus zur Verfügung:

```
begin
   lese ε, δ und n;
   bestimme s als Funktion von ε, δ, n;
   for i = 1 to s do
      fordere ein Beispiel (x_i, b_i) an;    /* b_i = 1 genau dann, wenn x_i ∈ Σ^{≤n}
                                                zum zu lernenden Konzept gehört */
   wähle eine Hypothese h_c ∈ H;
end.
```

Ein Lernalgorithmus L wird versuchen, eine konsistente Hypothese h zu finden. (Eine Hypothese h heißt genau dann konsistent, wenn sie auf den Beispielen das gleiche Verhalten wie das zu lernende Konzept zeigt; $x_i \in h \Leftrightarrow b_i = 1$.) Das Konsistenzproblem ist das Problem der Konstruktion einer konsistenten Hypothese.

Wann ist ein Lernalgorithmus L akzeptabel? L muß eine ausreichend große Menge von Beispielen anfordern, so daß konsistente Hypothesen eine gute Approximation des zu lernenden Konzeptes sind. Aber der Lernalgorithmus muß auch effizient sein: Die Beispielsmenge darf nicht zu groß sein, und das Konsistenzproblem muß effizient lösbar sein.

Zuerst formalisieren wir den Begriff „einer guten Approximation“.

1.4.1 Definition **a)** Die symmetrische Differenz für $X, Y \subseteq \Sigma^*$ definieren wir durch

$$X \oplus Y := (X \cup Y) \setminus X \cap Y.$$

b) Für ein Konzept $c \in \mathcal{C}$, eine Hypothese $h_c \in \mathcal{H}$ und eine Verteilung D für $\Sigma^{\leq n}$ definieren wir den Fehler als

$$Fehler_D(c, h_c) = \sum_{x \in c \oplus h_c} D(x).$$

Beachte, daß wir den Fehler mit der gleichen Verteilung D definieren, die auch die Erzeugung der Beispiele „steuert“ (die sogenannte FAIRNESS Bedingung): Wir sollten den Lernalgorithmus nicht bestrafen, wenn eine Verteilung „schlechte“ Beispiele (zum Beispiel nur negative Beispiele) liefert. Statt dessen sollte sich der Lernalgorithmus auf die Verteilung einstellen (und im obigen Fall die leere Hypothese produzieren).

Sei D eine vorgegebene unbekannte Verteilung. Die Hypothese h_c wird zu einer Zufallsvariablen, denn die Wahl von h_c hängt von den zufällig produzierten Beispielen ab. Sollten wir verlagen, daß $Fehler_D(c, h_c)$ für alle möglichen Beispielsmengen klein ist? Eine solche Anforderung wäre unfair, denn selbst eine „gute“ Verteilung kann (mit kleiner Wahrscheinlichkeit) schlechte Beispielmengen liefern. Statt dessen verlangen wir, daß

$$Fehler_D(c, h_c) \text{ mit hoher Wahrscheinlichkeit klein ist.}$$

Zu einer Formalisierung führen wir den Fehlerparameter ε und den Vertrauensparameter δ ein.

1.4.2 Definition Der Lernalgorithmus L ist genau dann ein PAC-Algorithmus für eine Konzeptklasse $\mathcal{C}$, wenn für alle $\varepsilon \in (0,1]$, für alle $\delta \in (0,1]$, für alle $n \in \mathbb{N}$, für alle Konzepte $c \in \mathcal{C}$ und für alle Verteilungen D auf $\Sigma^{\leq n}$ gilt:

$$prob[Fehler_D(c, h_c) \leq \varepsilon] \geq 1 - \delta.$$

PAC-Algorithmen müssen sich also **jeder** Verteilung anpassen. Wir werden später Konzeptklassen kennenlernen, die für bestimmte Verteilungen leichter zu erlernen sind.

Beispiel Ein PAC-Algorithmus für die Konzeptklasse $Monom_n$ (ohne Analyse).
ALGORITHMUS:

- Lies ε, δ, und n, und bestimme $s(\varepsilon, \delta, n)$ als die Anzahl der anzufordernden Beispiele.
- Erste Hypothese
$$x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \ldots \wedge x_n \wedge \neg x_n.$$
- Bestimme unter den Beispielen die positiven Beispiele;
dies seien die Beispiele $y_1, \ldots, y_r \in \{0,1\}^n$.
- For $i := 1$ to r
  for $j := 1$ to n
    wenn $y_{ij} = 1$ dann entferne $\neg x_j$
    wenn $y_{ij} = 0$ dann entferne x_j
(Hierbei ist y_{ij} die j-te Position im i-ten Beispiel.)

Wir beweisen später, daß wir einen PAC-Algorithmus erhalten, falls die Anzahl $s(\varepsilon, \delta, n)$ der angeforderten Beispiele die Ungleichung
$$s(\varepsilon, \delta, n) = \Omega(\frac{1}{\varepsilon}\ln(\frac{1}{\delta}) + \frac{n}{\varepsilon})$$
erfüllt.

Wir wissen jetzt, was ein PAC-Algorithmus ist. Um die Qualität eines PAC-Algorithmus zu klären, müssen wir seine Zeiteffizienz bestimmen. Deshalb betrachten wir

- die Anzahl der angeforderten Beispiele, sowie
- die Zeit, die für die Durchführung des Interpolationsschritts (also die Bestimmung der Hypothese h_c) benötigt wird.

1.4.3 Definition **a)** Sei L ein PAC-Algorithmus für die Konzeptklasse $\mathcal{C}$. Für $\varepsilon \in (0,1], \delta \in (0,1]$ und $n \in \mathbb{N}$ sei
$$s_L(\varepsilon, \delta, n)$$
die von L angeforderte Anzahl an Beispielen für Fehlerparameter ε, Vertrauensparameter δ und Beispiellänge n, als obere Schranke für die Anzahl der Beispiele.
b) Die Beispielsanzahl von L ist definiert durch:
$$Beispiel(\mathcal{C})(\varepsilon, \delta, n) = \min\{s_L(\varepsilon, \delta, n) \mid L \text{ ist ein PAC-Algorithmus für } \mathcal{C}\}.$$

$Beispiel(\mathcal{C})(\varepsilon, \delta, n)$ gibt somit die minimale Anzahl an benötigten Beipielen für n, Fehler ε und Vertrauenswert δ an.

Im nächsten Kapitel werden wir für viele Konzeptklassen die Beispielsanzahl zumindest annähernd bestimmen. Als nächstes betrachten wir die vom Algorithmus benötigte Zeit.

1.4.4 Definition Sei L ein PAC-Algorithmus für die Konzeptklasse $\mathcal{C}$. Sei $Name : \mathcal{C} \to \mathcal{P}(\{0,1\}^*)$ eine Namensfunktion für $\mathcal{C}$. Für jedes Konzept $c \in \mathcal{C}$ definieren wir die Länge des kürzesten Namens als

$$länge(c) = \min\{|w| \mid w \in name(c)\}.$$

a) Dann bezeichnet

$$zeit_L(\varepsilon, \delta, n, l)$$

das Maximum der von L benötigten Schrittzahl über alle Konzepte $c \in \mathcal{C}$ mit $länge(c) \leq l$, über alle Verteilungen D auf $\Sigma^{\leq n}$ und über alle Beispielsmengen.
b) Die Konzeptklasse $\mathcal{C}$, mit Namensfunktion $Name : \mathcal{C} \to \mathcal{P}(\{0,1\}^*)$ heißt genau dann lernbar in Zeit höchtens $zeit(\varepsilon, \delta, n, l)$, wenn es einen PAC-Algorithmus L für $\mathcal{C}$ gibt mit

$$zeit(\varepsilon, \delta, n, l) \geq \ zeit_L(\varepsilon, \delta, n, l).$$

Für uns sind natürlich diejenigen Konzeptklassen besonders wichtig, die effiziente Lernalgorithmen besitzen. Was heißt hierbei „effizient“?

1.4.5 Definition Sei $\mathcal{C}$ eine Konzeptklasse.
a) Wir sagen, daß $\mathcal{C}$ polynomielle Beispielsanzahl besitzt, falls

$$Beispiel(\mathcal{C})(\varepsilon, \delta, n) \leq poly_1(\frac{1}{\varepsilon}, \frac{1}{\delta}, n)$$

für alle ε, δ und n gilt.
($poly_1$ bezeichnet ein Polynom in drei Veränderlichen.)
b) $\mathcal{C}$, mit Namensfunktion $Name$ heißt polynomiell lernbar, falls $\mathcal{C}$ mit $Name$ in Zeit höchstens

$$poly_2(\frac{1}{\varepsilon}, \frac{1}{\delta}, n, l)$$

im PAC-Sinne erlernbar ist.
($poly_2$ bezeichnet ein Polynom in vier Veränderlichen.)

Ein zentrales Problem der Vorlesung ist die Bestimmung der polynomiell lernbaren Konzeptklassen. Wir werden zeigen, daß -bei geeigneter Wahl der Hypothensenklasse- zum Beispiel die folgenden Konzeptklassen polynomiell lernbar sind:

- $Monom_n$
- k-CNF_n und k-DNF_n
- k-Klauseln CNF_n und k-Term DNF
- $Threshold_n$

- Sym_n
- $List_n$

Wir lernen „beinahe“ effiziente Algorithmen für $Decision_n$ kennen. Für DNF_n werden wir ebenfalls einen „beinahe“ effizienten Lernalgorithmus beschreiben; allerdings fordern wir hier die Gleichverteilung (wir betrachten also das verteilungs-spezifische Lernmodell).

Weiterhin sammeln wir Beweise für nicht-polynomielle Lernbarkeit. Dies betrifft das Lernen von k-Term DNF durch k-Term DNF, sowie das Lernen von endlichen Automaten und Threshold-Schaltkreisen konstanter Tiefe durch beliebige Hypothesenklassen.

Damit erhalten wir auch ein Beispiel, bei dem die Wahl der Hypothesenklasse wesentlich für ein effizientes Lernen ist: Die Konzeptklasse k-Term DNF.

Zuletzt werden wir strukturelle Fragen behandeln. Hierzu gehören folgende Fragen:

- Läßt sich ein nicht zu großer Fehler stets verringern?
- Genügt es für das erfolgreiche Lernen, eine Hypothese zu finden, die sehr viel kürzer als die Konkatentation aller Beispiele ist?

1.5 Aktives Lernen

Wir betrachten ein Lernmodell mit „Elementfragen“ (gehört Beispiel x zum Konzept?) und „Äquivalenzfragen“ (gib, wenn möglich, ein Gegenbeispiel zur gegenwärtigen Hypothese).

Wir werden sehen, daß durch dieses Lernmodell die (bisher nicht effizient lernbare) Konzeptklasse der deterministischen endlichen Automaten effizient lernbar wird: Ein aktiver Student lernt leichter!

1.6 Wichtige Ungleichungen

In diesem Abschnitt stellen wir Ungleichungen vor, die wir in der Vorlesung häufig benötigen.

Bemerkung (1) Für $-1 < \varepsilon < \infty$ gilt: $\ln(1+\varepsilon) \leq \varepsilon$.
(2) Für $-\infty < \varepsilon < \infty$ gilt: $1+\varepsilon \leq e^{\varepsilon}$.
(3) Es gilt: $(1+\frac{1}{n})^n \leq e$.
(4) Die Chernoff-Schranken besagen: Wenn $Y_1,\ldots,Y_s \in \{0,1\}$ unabhängige Zufallsvariablen sind, und $p = prob[Y_1 = 1] = prob[Y_2 = 1] = \ldots = prob[Y_s = 1]$ ist, dann gilt für $0 \leq \alpha \leq 1$:

$$prob[\sum_{i=1}^{s} Y_i \geq (1+\alpha)p \cdot s] \leq e^{-\frac{\alpha^2 s \cdot p}{3}}$$

sowie

$$prob[\sum_{i=1}^{s} Y_i \leq (1-\alpha)p \cdot s] \leq e^{-\frac{\alpha^2 s \cdot p}{2}}.$$

(Das heißt, ein großer Verstoß gegen den Erwartungswert $s \cdot p$ ist unwahrscheinlich.)

Beweis (Ohne Beweis von (4).)
(1) Sei $\varepsilon \geq 0$. Für die Funktion ln gilt, daß die Steigung der Sekante durch Punkt 1 und $1+\varepsilon$ (bzw. $1-\varepsilon$ und 1) kleiner oder gleich (bzw. größer oder gleich) der Tangentensteigung in Punkt 1 ist. Damit erhalten wir:

$$\underbrace{\frac{\ln(1+\varepsilon)-\ln(1)}{(1+\varepsilon)-1}}_{\text{Sekantensteigung von } 1,1+\varepsilon} = \frac{\ln(1+\varepsilon)}{\varepsilon} \leq \underbrace{1}_{\text{Tangentensteigung in } 1}$$

$$\Rightarrow \qquad \ln(1+\varepsilon) \leq \varepsilon.$$

Für $0 \leq \varepsilon < 1$ gilt:

$$\frac{\ln(1)-\ln(1-\varepsilon)}{1-(1-\varepsilon)} = \frac{-\ln(1-\varepsilon)}{\varepsilon} \geq 1$$

$$\Rightarrow \qquad \ln(1-\varepsilon) \leq -\varepsilon.$$

(2) Wir benutzen (1). Für $\varepsilon \geq 0$ ist $\ln(1+\varepsilon) \leq \varepsilon$ und damit $1+\varepsilon \leq e^{\varepsilon}$.
Für $0 \leq \varepsilon < 1$ ist $\ln(1-\varepsilon) \leq -\varepsilon$. Also folgt $1-\varepsilon \leq e^{-\varepsilon}$. Für $\varepsilon \geq 1$ ist $1-\varepsilon \leq 0$, aber $e^{-\varepsilon} \geq 0$ und damit gilt (2) für alle Werte von ε.

(3) Mit (2) folgt: $(1+\frac{1}{n})^n \leq (e^{\frac{1}{n}})^n = e$. □

2 Beispielsanzahl und die Vapnik-Chervonenkis Dimension

Wir beschäftigen uns mit dem folgenden Problem: Wieviele Beispiele benötigt ein Lernalgorithmus, um eine vorgegebene Konzeptklasse $\mathcal{C}$ über Σ im PAC-Sinne zu erlernen. Die benötigte Anzahl hängt natürlich vom Vertrauensparameter δ und vom Fehlerparameter ε ab: Je kleiner ε und δ umso größer ist die benötigte Anzahl. Die Konzeptklasse $\mathcal{C}$ ist der dritte, aber wesentlichste Faktor in der Bestimmung der Beispielsanzahl.

2.1 Die Vapnik-Chervonenkis Dimension

Gegeben sei eine Konzeptklasse $\mathcal{C}$ über Σ. Intuitiv ist zu erwarten, daß die benötigte Beispielsanzahl von der „Anzahl der Freiheitsgrade“ von $\mathcal{C}$ abhängt. Wie sollen wir Freiheitsgrade messen?

2.1.1 Definition **a)** Sei $\mathcal{C}$ eine Konzeptklasse über Σ und $S \subseteq \Sigma^*$. Wir sagen, daß $\mathcal{C}$ die Menge S zertrümmert, falls

$$\{S \cap c \mid c \in \mathcal{C}\} = \mathcal{P}(\mathcal{S})$$

ist.
b) Wir definieren die Vapnik-Chervonenkis Dimension von $\mathcal{C}$ (Notation: $VC(\mathcal{C})$) durch

$$VC(\mathcal{C}) = \max\{|S| \mid \mathcal{C} \text{ zertrümmert } S\}.$$

Wir sehen im Laufe dieses Kapitels, daß $VC(\mathcal{C})$ tatsächlich eine akzeptable Definition der Anzahl an Freiheitsgraden liefert. Vorest überprüfen wir dies exemplarisch an Hand der nächsten beiden Beispiele:

Beispiele

1. Wir betrachten die Konzeptklasse $Rechteck_{n,2}$. Jedes Konzept in $Rechteck_{n,2}$ wird durch zwei Punkte (x_1, y_1) und $(x_2, y_2) \in \mathbb{Z}^2$ (mit $-2^n \leq x_1, x_2, y_1, y_2 \leq 2^n$), wobei $x_1 \leq x_2$ und $y_1 \leq y_2$ ist, bestimmt. Das Konzept zu den Punkten (x_1, y_1), (x_2, y_2) besteht aus allen Punkten (mit ganzzahligen Koordinaten) des von (x_1, y_1) und (x_2, y_2) aufgespannten achsenparallelen Rechtecks. Intuitiv hat $Rechteck_{n,2}$ vier Freiheitsgrade: Seine vier Kanten sind zu bestimmen.

   Wir zeigen zuerst, daß $VC(Rechteck_{n,2}) \geq 4$ ist. Dazu betrachten wir die vier Punkte $e_1 = (-1, 0)$, $e_2 = (0, 1)$, $e_3 = (0, -1)$ und $e_4 = (1, 0)$ und setzen $S = \{e_1, e_2, e_3, e_4\}$. Es ist leicht zu überprüfen, daß wir alle 16 Teilmengen von S als Schnitt von S mit dem Konzept eines Rechtecks aus $Rechteck_{n,2}$ (für $n \geq 1$) erhalten. (Die Teilmenge $T = \{e_1, e_2, e_3\}$ von S erhalten wir z. B. durch den Schnitt von S mit dem Konzept von $R_1 \simeq ((-1, 1), (0, -1))$:

$$T = \underbrace{\{(-1,0),(0,1),(0,-1),(1,0)\}}_{S} \cap \underbrace{\{(-1,1),(0,1),(-1,0),(0,0),(-1,-1),(0,-1)\}}_{\text{Konzept zu } R_1}.)$$

   Es bleibt noch zu zeigen, daß $VC(Rechteck_{n,2}) \leq 4$ ist. Angenommen $Rechteck_{n,2}$ zertrümmert eine Menge S der Mächtigkeit 5. Wir bestimmen die Punkte p_1 (p_2) mit minimaler (maximaler) x-Koordinate und die Punkte p_3 (p_4) mit minimaler (maximaler) y-Koordinate aus S. Wir versuchen die Teilmenge $T = \{p_1, p_2, p_3, p_4\}$ von S zu erzeugen. Dazu betrachten wir das kleinste achsenparallele Rechteck R, das alle Punkte aus T enthält. Nach Wahl von T enthält R dann aber alle Punkte aus S. Dies bedeutet, daß die Teilmenge T nicht als der Durchschnitt von S mit einem Konzept aus $R_{n,2}$ erhalten werden kann. $R_{n,2}$ zertrümmert die Menge S also nicht!

   Allgemein definieren wir die Konzeptklasse $R_{n,d}$ wie folgt: Ein Konzept aus $R_{n,d}$ wird durch ein achsenparalleles Rechteck in $\mathbb{R}^d$ spezifiziert und besteht aus allen

Punkten $(x_1, \ldots, x_d) \in \mathbb{Z}^d$ (mit $-2^n \leq x_i \leq 2^n$ für $1 \leq i \leq d$) die in dem Rechteck enthalten sind. In den Übungen (Aufgabe 3.1 a)) werden wir sehen, daß

$$VC(R_{n,d}) = 2d$$

ist.

2. Als nächstes Beispiel betrachten wir die Konzeptklasse $Boolean_n$. $Boolean_n$ besitzt für jede Teilmenge von $\{0,1\}^n$ ein Konzept. Wir behaupten, daß

$$VC(\mathrm{Boolean_n}) = 2^{\mathrm{n}}$$

gilt. $Boolean_n$ zertrümmert die Menge $S = \{0,1\}^n$, denn $Boolean_n$ besitzt für jede Teilmenge von S ein Konzept. Andererseits ist S auch die größte von $Boolean_n$ zertrümmerte Menge: Eine Menge der Mächtigkeit $> 2^n$ muß mindestens ein Element aus dem Komplement von $\{0,1\}^n$ besitzen; dieses kann nicht isoliert werden, da die Konzepte aus $Boolean_n$ nur Elemente aus $\{0,1\}^n$ enthalten.

   2^n ist auch eine „natürliche“ Festsetzung für die Anzahl der Freiheitsgrade von $Boolean_n$: Ein Konzept aus $Boolean_n$ kann erst dann eindeutig bestimmt werden, wenn sein Verhalten auf allen 2^n Elementen von $\{0,1\}^n$ bekannt ist.

Wir beginnen unsere Untersuchung der VC-Dimension mit einigen elementaren Eigenschaften:

2.1.2 Satz **a)** *Seien $\mathcal{C}_1$ und $\mathcal{C}_2$ zwei Konzeptklassen über Σ mit $\mathcal{C}_1 \subseteq \mathcal{C}_2$. Dann gilt*

$$VC(\mathcal{C}_1) \leq VC(\mathcal{C}_2).$$

b) *Für eine Konzeptklasse $\mathcal{C}$ sei die Komplementkonzeptklasse $\bar{\mathcal{C}}$ definiert durch $\bar{\mathcal{C}} = \{\bar{c} \mid c \in \mathcal{C}\}$, wobei $\bar{c} = \Sigma^* \setminus c$ ist. Dann gilt*

$$VC(\mathcal{C}) = VC(\bar{\mathcal{C}}).$$

c) *Für jede Konzeptklasse $\mathcal{C}$ gilt*

$$VC(\mathcal{C}) \leq \lfloor \log_2 |\mathcal{C}| \rfloor.$$

Beweis a) und b) werden in den Übungen behandelt (Aufgabe 2.1 a),b)).
c) folgt aus der Aussage, daß $2^{VC(\mathcal{C})} \leq |\mathcal{C}|$ ist. Diese Aussage wird in Aufgabe 1.3 a) bewiesen. □

Als nächstes bestimmen wir die VC-Dimension der wichtigsten Konzeptklassen zumindest asymptotisch:

2.1.3 Satz *Es gelten die folgenden Aussagen:*

a) $VC(Monoton\text{-}Monom_n) = n$,

b) $VC(Monom_n) = \Theta(n)$,

c) $\binom{n}{k} \leq VC(k\text{-}CNF_n) = VC(k\text{-}DNF_n) \leq \binom{n}{k} \cdot 2^k$,
wenn k fest ist, folgt (für wachsendes n) $VC(k\text{-}CNF_n) = VC(k\text{-}DNF_n) = \Theta(n^k)$,

d) $VC(k\text{-}Klauseln\ CNF_n) = VC(k\text{-}Term\ DNF_n) = \Theta(k \cdot n)$, *für* $k \leq 2^{n/2}$,

e) $VC(CNF_n) = VC(DNF_n) = VC(Boolean_n) = 2^n$,

f) $VC(Threshold_n) = n + 1$,

g) $VC(Sym_n) = n + 1$,

h) $n - 1 \leq VC(Aut_{n,\Sigma}) = O(n \cdot |\Sigma| \cdot \log_2 n)$,

i) $VC(Rechteck_{n,d}) = 2d$.

Beweis a), b), c), d), g) und i) werden in den Übungen gezeigt (Aufgabe 2.3, Aufgabe 1.3 b), Aufgabe 2.4 a)-c), Aufgabe 3.2, Aufgabe 2.2 und Aufgabe 3.1 a)).
e) Wir haben bereits gezeigt, daß $VC(Boolean_n) = 2^n$ gilt. Die Behauptung folgt, da $CNF_n = DNF_n = Boolean_n$ ist. □

f) Wir zeigen zuerst $VC(Threshold_n) \leq n+1$. Hierfür benötigen wir den Satz von Radon, den wir ohne Beweis anwenden. Der Satz von Radon besagt, daß es für jede Menge $S \subseteq \mathbb{R}^n$ mit $|S| \geq n + 2$, eine nicht-leere Teilmenge $T \subset S$ gibt, so daß

$$\text{konvexe Hülle}(T) \cap \text{konvexe Hülle}(S \setminus T) \neq \emptyset \qquad (*)$$

gilt. [Eine Menge Y heißt genau dann konvex, wenn für je zwei Punkte aus Y auch die verbindende Gerade in Y liegt. Die konvexe Hülle einer Menge X ist definiert als die kleinste konvexe Menge, die die Menge X enthält.]

Angenommen $Threshold_n$ zertrümmert eine Menge S von $n + 2$ Punkten. Mit dem Satz von Radon erhalten wir eine nicht-leere Teilmenge $T \subset S$ mit der Eigenschaft $(*)$. Da $Threshold_n$ die Menge S zertrümmert, gibt es ein Konzept c mit $S \cap c = T$. Das Konzept c besteht aus allen Punkten $x \in \{0,1\}^n$ mit $\sum_{i=1}^{n} w_i x_i \geq t$ (für reelle Zahlen $w_1, \ldots, w_n$ und t). Die Menge $H = \{x \mid x \in \mathbb{R}^n, \sum_{i=1}^{n} w_i x_i = t\}$ bildet eine Hyperebene, die $\mathbb{R}^n$ in die beiden Halbräume

$$H \geq = \{x \mid x \in \mathbb{R}^n, \sum_{i=1}^{n} w_i x_i \geq t\} \text{ (mit } c \subset H_{\geq})$$

und

$$H_< = \{x \mid x \in \mathbb{R}^n, \sum_{i=1}^{n} w_i x_i < t\} \text{ (mit } c \cap H_< = \emptyset)$$

zerschneidet. Diese beiden Halbräume sind konvexe Mengen: Mit je 2 Punkten ist auch die verbundene Gerade in der Menge enthalten. Wir erhalten

$$T \subseteq c \subseteq H_\geq$$

und damit

$$\text{konvexe Hülle}(T) \subseteq \text{konvexe Hülle}(H_\geq) = H_\geq,$$

beziehungsweise

$$S \setminus T \subseteq \bar{c} \subseteq H_<$$

und damit

$$\text{konvexe Hülle}(S \setminus T) \subseteq \text{konvexe Hülle}(H_<) = H_<.$$

Es gilt

$$\text{konvexe Hülle}(T) \cap \text{konvexe Hülle}(S \setminus T) \subseteq H_\geq \cap H_< = \emptyset$$

im WIDERSPRUCH zum Satz von Radon.

Wir weisen jetzt $VC(Threshold_n) = n + 1$ nach indem wir zeigen, daß $Threshold_n$ eine Menge S der Mächtigkeit $n + 1$ zertrümmert. Wir wählen $S = \{0, e_1, \ldots, e_n\}$, wobei 0 der Nullvektor ist und e_i der Vektor ist, der eine 1 in Komponente i und ansonsten nur Nullen besitzt. Sei T eine beliebige Teilmenge von S.
Fall 1: $0 \notin T$
Wähle das Konzept c, das der Thresholdfunktion mit Gewichten

$$w_i = \begin{cases} 1 & \text{wenn } e_i \in T \\ -1 & \text{sonst} \end{cases}$$

und dem Schwellenwert $t = \frac{1}{2}$ entspricht.
Fall 2: $0 \in T$
Wähle das Konzept c, das der Thresholdfunktion mit den im Fall 1 definierten Gewichten und dem Schwellenwert $t = 0$ entspricht.

In beiden Fällen läßt sich leicht verifizieren, daß $S \cap c = T$ gilt. □

h) Wir weisen zuerst $n \leq VC(Aut_{n,\Sigma})$ nach. Durch Umbenennung der Buchstaben in Σ, $|\Sigma| \geq 2$, können wir erreichen, daß $\{0, 1\} \subseteq \Sigma$ gilt. Nach dieser Umbenennung erhalten wir die Inklusion $Monom_{n-1} \subseteq Aut_{n,\Sigma}$, denn jedes Monom-Konzept (mit höchstens $n-1$ Literalen) kann durch einen Automaten mit n Zuständen erkannt werden. Somit gilt

$$n - 1 \leq VC(Aut_{n,\Sigma}).$$

Zum Beweis der oberen Schranke $VC(Aut_{n,\Sigma}) = O(n \cdot |\Sigma| \cdot \log_2 n)$ werden wir anwenden, daß $VC(\mathcal{C}) \leq \log_2 |\mathcal{C}|$ gilt. Sei a_n die Anzahl der deterministischen endlichen Automaten

mit genau n Zuständen. Wir groß ist a_n?
Für jeden der n Zustände müssen $|\Sigma|$-viele Nachfolgezustände erklärt werden. Dafür gibt es $n^{|\Sigma|}$ viele Möglichkeiten. Als Konsequenz erhalten wir $a_n \leq n^{|\Sigma| \cdot n}$.
Somit erhalten wir

$$\begin{array}{rcl} |Aut_{n,\Sigma}| & \leq & a_1 + a_2 + \ldots + a_n \quad \leq \quad n \cdot a_n = n^{|\Sigma| \cdot n+1} \\ \Rightarrow \; VC(Aut_{n,\Sigma}) & \leq & (|\Sigma| \cdot n + 1) \cdot log_2 n \quad = \quad O(n \cdot |\Sigma| \cdot log_2 n). \end{array}$$

■

Wir schließen die Betrachtung der VC-Dimension mit zwei technischen Lemmas ab, die im wesentlichen klären wie unscharf die Abschätzung $VC(\mathcal{C}) \leq log_2|\mathcal{C}|$ sein kann.

2.1.4 Lemma Sei $\mathcal{C}$ eine Konzeptklasse über Σ. Für eine Teilmenge X von $\sum^*$ sei $c \subseteq X$ für jedes Konzept $c \in \mathcal{C}$. Dann gilt

$$|\mathcal{C}| \leq (|X| + 1)^{VC(\mathcal{C})}.$$

(Also gilt für $X = \{0,1\}^{\leq n}$ die Ungleichung $|\mathcal{C}| \leq 2^{(n+1)VC(\mathcal{C})}$.)

Beweis Wir führen den Beweis durch Induktion über die Größe von $|X|$. Für $|X| = 1$ kann es höchstens zwei Konzepte geben, und zwar X und $\emptyset$. Im Fall $|\mathcal{C}| = 1$ ist zu zeigen, daß $1 \leq 2^{VC(\mathcal{C})}$ gilt. Da mit einem Konzept nur die leere Menge zertrümmert werden kann, ist $VC(\mathcal{C}) = 0$ und die Behauptung richtig. Im Fall $|\mathcal{C}| = 2$ ist zu zeigen, daß $2 \leq 2^{VC(\mathcal{C})}$ gilt. Mit den Mengen X und $\emptyset$ kann jede ein-elementige Teilmenge von X zertrümmert werden und $VC(\mathcal{C}) = 1$ folgt. Die Behauptung ist also auch in diesem Fall richtig.

Die Behauptung sei für $|X| = k$ bewiesen.

$k \leadsto k+1$
Es gelte $|X| = k + 1$. Sei x ein beliebiges Element aus X. Wir betrachten die Konzeptklassen

$$\mathcal{C}_1 = \{c \mid c \in \mathcal{C},\ x \notin c \text{ und } c \cup \{x\} \in \mathcal{C}\}$$

und

$$\mathcal{C}_2 = \mathcal{C} \setminus \mathcal{C}_1.$$

Behauptung $\mathcal{C}_1$ ist eine Konzeptklasse über $X \setminus \{x\}$ und $VC(\mathcal{C}_1) \leq VC(\mathcal{C}) - 1$.

Beweis Angenommen $VC(\mathcal{C}_1) = VC(\mathcal{C})$. Dann zertrümmert $\mathcal{C}_1$ eine Menge $S \subset X - \{x\}$ mit $|S| = VC(\mathcal{C})$. Das bedeutet aber, daß $\mathcal{C}$ die Menge $S \cup \{x\}$ zertrümmert, denn: Wähle $T \subseteq S \cup \{x\}$.
Fall 1: $x \notin T$. Dann gibt es $c \in \mathcal{C}_1$ mit $S \cap c = T$ und $(S \cup \{x\}) \cap c = T$ folgt.
Fall 2: $x \in T$. Dann gibt es ein Konzept $c \in \mathcal{C}_1$ mit $S \cap c = T \setminus \{x\}$. Aber das Konzept $d = c \cup \{x\}$ gehört zu $\mathcal{C}$ und es gilt $(S \cup \{x\}) \cap d = T$.

Da $|S \cup \{x\}| = VC(\mathcal{C}) + 1$, kann die Menge $S \cup \{x\}$ aber nicht zertrümmert werden und wir haben einen Widerspruch zu der Annahme $VC(\mathcal{C}_1) = VC(\mathcal{C})$ erhalten. □

Behauptung $\mathcal{C}_2$ kann als Konzeptklasse über $X \setminus \{x\}$ aufgefaßt werden, d.h. $|\mathcal{C}_2| = |\{c \setminus \{x\} \mid c \in \mathcal{C}_2\}|$.

Beweis Seien c und d beliebige, aber verschiedene Konzepte aus $\mathcal{C}_2$. Es genügt zu zeigen, daß dann $c \setminus \{x\} \neq d \setminus \{x\}$ ist. Sei anderenfalls $c \setminus \{x\} = d \setminus \{x\}$ und o.B.d.A. $c = d \cup \{x\}$. Dann ist $d \cup \{x\} \in \mathcal{C}$ und $x \notin d$. Dies ist unmöglich, da dann $d \in \mathcal{C}_1$ ist und nicht zu $\mathcal{C}_2$ gehört, wie angenommen. $\square$

Wir können also $\mathcal{C}_1$ und $\mathcal{C}_2$ als Konzeptklassen über $X \setminus \{x\}$ auffassen und damit die Induktionsvoraussetzung anwenden:

$$\begin{aligned}
|\mathcal{C}| &= |\mathcal{C}_1| + |\mathcal{C}_2| \\
&= |\mathcal{C}_1| + |\{c \setminus \{x\} \mid c \in \mathcal{C}_2\}| \\
&\underset{\text{Ind.Vor.}}{\leq} (|X| - 1 + 1)^{VC(\mathcal{C})-1} + (|X| - 1 + 1)^{VC(\mathcal{C})} \\
&= (|X|)^{VC(\mathcal{C})-1} + (|X|)^{VC(\mathcal{C})} \\
&= (|X|)^{VC(\mathcal{C})-1} \cdot (1 + |X|) \\
&\leq (|X| + 1)^{VC(\mathcal{C})-1} \cdot (1 + |X|) \\
&= (|X| + 1)^{VC(\mathcal{C})}.
\end{aligned}$$

$\blacksquare$

Wir verschärfen nun die Aussage von Lemma 2.1.4; für größere VC-Dimensionen ist das folgende Lemma eine bessere Abschätzung als Lemma 2.1.4:

2.1.5 Lemma Sei $\mathcal{C}$ eine Konzeptklasse über Σ. Für eine Teilmenge X von $\sum^*$ sei $c \subseteq X$ für jedes Konzept $c \in \mathcal{C}$. Dann gilt für $VC(\mathcal{C}) \geq 1$:

$$|\mathcal{C}| \underset{I}{\leq} \Phi(|X|, VC(\mathcal{C})) \underset{II}{\leq} \left(\frac{e \cdot |X|}{VC(\mathcal{C})} \right)^{VC(\mathcal{C})},$$

wobei $\Phi(m, d) = \sum_{i=0}^{d} \binom{m}{i}$ ist.

Beweis I: Wir benutzen die Abkürzung $d = VC(\mathcal{C})$.

In Lemma 2.1.4 haben wir gezeigt, daß $|\mathcal{C}| \leq |\mathcal{C}_1| + |\mathcal{C}_2|$ gilt. Wir haben auch gesehen, daß $VC(\mathcal{C}_1) \leq d - 1$, $\mathcal{C}_1$ eine Konzeptklasse über $X \setminus \{x\}$ und $|\mathcal{C}_2| = |\{c \setminus \{x\} | c \in \mathcal{C}_2\}|$ ist.

Auch hier führen wir einen induktiven Beweis über $|X|$. Für $|X| = 1$ ist die Aussage korrekt.

Die Behauptung sei für $|X| = m-1$ bewiesen.

$m-1 \rightsquigarrow m$:
Es gelte $|X| = m$. Dann ist

$$\begin{aligned}
|\mathcal{C}| &\leq |\mathcal{C}_1| + |\{c \setminus \{x\} | c \in \mathcal{C}_2\}| \\
&\underset{\text{Ind.vor.}}{\leq} \Phi(m-1,d) + \Phi(m-1,d-1) \\
&= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} \\
&= \binom{m-1}{0} + \sum_{i=0}^{d-1} \left(\binom{m-1}{i+1} + \binom{m-1}{i} \right).
\end{aligned}$$

Es gilt:

$$\begin{aligned}
\binom{m-1}{i+1} + \binom{m-1}{i} &= \binom{m-1}{i} \frac{m-(i+1)}{i+1} + \binom{m-1}{i} \\
&= \binom{m-1}{i} \left(\frac{m-(i+1)}{i+1} + 1 \right) \\
&= \binom{m-1}{i} \left(\frac{m-(i+1)}{i+1} + \frac{i+1}{i+1} \right) \\
&= \binom{m-1}{i} \left(\frac{m}{i+1} \right) \\
&= \binom{m}{i+1}.
\end{aligned}$$

Deshalb folgt:

$$|\mathcal{C}| \leq \binom{m-1}{0} + \sum_{i=0}^{d-1} \binom{m}{i+1} = \sum_{i=0}^{d} \binom{m}{i}.$$

□

II: Der Nachweis der Ungleichung $\Phi(m,d) \leq \left(\frac{e \cdot m}{d}\right)^d$ erfolgt durch Induktion über m. Für $m = 1$ gibt es höchstens zwei Konzepte. Für $|\mathcal{C}| = 1$ ist $VC(\mathcal{C}) = 0$ und dieser Fall ist in der Behauptung ausgeklammert. Für $|\mathcal{C}| = 2$ ist $VC(\mathcal{C}) = 1$ und es ist zu zeigen, daß $\binom{1}{0} + \binom{1}{1} = 2 \leq \frac{e^2}{2}$ gilt. Dies ist aber offensichtlich, da $e \geq 2$.

Die Behauptung sei für $|X| = m-1$ bewiesen.

$m-1 \rightsquigarrow m$:

Es gilt (unter Benutzung der Ungleichungen von Paragraph 1.5):

$$\begin{aligned}
\Phi(m,d) &= \Phi(m-1,d-1)+\Phi(m-1,d)\\
&\overset{\text{Ind.Vor.}}{\leq} \left(\tfrac{e(m-1)}{d-1}\right)^{d-1}+\left(\tfrac{e(m-1)}{d}\right)^{d}\\
&= \left(\tfrac{e\cdot m}{d}\right)^{d}\left(\tfrac{1}{e}\tfrac{(m-1)^{d-1}}{m^d}\tfrac{d^d}{(d-1)^{d-1}}+\tfrac{(m-1)^d}{m^d}\right)\\
&= \left(\tfrac{e\cdot m}{d}\right)^{d}\left(\tfrac{d}{m}\cdot\left(1-\tfrac{1}{m}\right)^{d-1}\cdot\tfrac{1}{e}\underbrace{\left(1+\frac{1}{d-1}\right)^{d-1}}_{\leq e}+\left(1-\tfrac{1}{m}\right)^{d}\right)\\
&\leq \left(\tfrac{e\cdot m}{d}\right)^{d}\underbrace{\left(1-\frac{1}{m}\right)^{d-1}}_{\leq e^{\frac{-(d-1)}{m}}}\underbrace{\left(\frac{d}{m}+1-\frac{1}{m}\right)}_{=1+\frac{d-1}{m}\leq e^{\frac{d-1}{m}}}\\
&\leq \left(\tfrac{e\cdot m}{d}\right)^{d}\cdot e^0.
\end{aligned}$$

■

2.2 Eine obere Schranke für die Beispielsanzahl

Es gelte $|\Sigma| = 2$. Sei $\mathcal{C}$ die zu lernende Konzeptklasse über Σ mit $VC(\mathcal{C}) = d$. Wir beschreiben jetzt einen PAC-Algorithmus für $\mathcal{C}$:

begin
  lies ε, δ und n;
  setze $s = \frac{1}{\varepsilon}(\ln(|\mathcal{C}|) + \ln(\frac{1}{\delta}))$;
  for $i := 1$ to s
    lies das i-te klassifizierte Beispiel x_i aus $\Sigma^{\leq n}$ mit Klassifikation $b_i \in \{0,1\}$;
  wähle ein **konsistentes** Konzept $h \in \mathcal{C}$ als Hypothese;
end.

2.2.1 Satz *Der obige Lernalgorithmus für $\mathcal{C}$ ist ein PAC-Algorithmus. Der Lernalgorithmus ist konsistent und erlernt $\mathcal{C}$ mit*

$$\frac{1}{\varepsilon}\left(\ln(|\mathcal{C}|)+\ln(\frac{1}{\delta})\right)\leq\frac{1}{\varepsilon}\left((n+1)\cdot VC(\mathcal{C})\cdot\ln(2)+\ln(\frac{1}{\delta})\right)$$

vielen Beispielen.

Beweis Wir müssen zeigen, daß

$$prob\ [\mathrm{Fehler_D(c, h_c)} \leq \varepsilon] \geq 1 - \delta$$

gilt, wobei h_c die vom Lernalgorithmus gewählte konsistente Hypothese ist. Dies ist für jedes Konzept $c \in \mathcal{C}$ und jede Wahrscheinlichkeitsverteilung D nachzuweisen. Seien also $c \in \mathcal{C}$ und D gegeben. Wir untersuchen zuerst, die Wahrscheinlichkeit, daß der Fehler zu groß ist:

$$\begin{aligned} p &= prob[Fehler_D(c, h_c) > \varepsilon] \\ &\leq prob[\text{ Es gibt eine konsistente Hypothese } h \text{ mit } Fehler_D(c, h) > \varepsilon] \\ &\leq \sum_{\substack{Fehler_D(c,h)>\varepsilon \\ h \in \mathcal{C}}} prob[h \text{ ist konsistent}]. \end{aligned}$$

Die letzte Ungleichung gilt, da

$prob[(\text{Hypothese } h_1 \text{ konsistent mit } Fehler_D(c, h_1) > \varepsilon) \cup (\text{ Hypothese } h_2 \text{ konsistent mit } Fehler_D(c, h_2) > \varepsilon)] \leq$
$prob[\text{ Hypothese } h_1 \text{ konsistent mit } Fehler_D(c, h_1) > \varepsilon] + prob[\text{ Hypothese } h_2 \text{ konsistent mit } Fehler_D(c, h_2) > \varepsilon]$
ist.

Wir betrachten dabei jeweils die Wahscheinlichkeit bei zufällig (gemäß D) gewählten Beispielsmengen. Wir wählen jetzt ein beliebiges Konzept $h \in \mathcal{C}$ mit $Fehler_D(c, h) > \varepsilon$. Dann ist die Wahrscheinlichkeit, daß ein zufällig gewähltes Beispiel konsistent mit h ist, durch $1 - \varepsilon$ nach oben beschränkt und somit

$$prob\ [h \text{ ist konsistent }] \leq (1 - \varepsilon)^s$$

$$\Rightarrow \quad p \leq \sum_{h \in \mathcal{C}} (1 - \varepsilon)^s = |\mathcal{C}| \cdot (1 - \varepsilon)^s.$$

s muß deshalb so groß gewählt werden, daß

$$|\mathcal{C}| \cdot (1 - \varepsilon)^s \leq \delta$$

gilt. Nachdem wir beide Seiten der Ungleichung logarithmieren, werden wir auf

$$\ln(|\mathcal{C}|) + s \cdot \underbrace{\ln(1 - \varepsilon)}_{\leq -\varepsilon} \leq \ln(\delta)$$

geführt. Somit reicht der Nachweis von

$$\begin{aligned} \ln(|\mathcal{C}|) - s \cdot \varepsilon &\leq \ln(\delta) \\ \Leftrightarrow \quad \ln(|\mathcal{C}|) - \ln(\delta) &\leq s \cdot \varepsilon \\ \Leftrightarrow \quad \tfrac{1}{\varepsilon}\Big(\ln(|\mathcal{C}|) + \ln(\tfrac{1}{\delta})\Big) &\leq s. \end{aligned}$$

Die zweite Ungleichung des Satzes folgt aus Lemma 2.1.4, da wir $|\Sigma| = 2$ gefordert haben und $c \subseteq \Sigma^{\leq n}$ annehmen können (da die Verteilung D nur Worten der Länge höchstens n eine positive Wahrscheinlichkeit gibt) und somit

$$\ln(|\mathcal{C}|) \leq \ln(2^{(n+1) \cdot VC(\mathcal{C})}) = (n + 1) \cdot VC(\mathcal{C}) \cdot \ln(2)$$

ist. □

Wir können jetzt Satz 2.1.3 und Satz 2.2.1 miteinander verbinden, um PAC-Algorithmen für viele wichtige Konzeptklassen zu erhalten:

2.2.2 Satz

a) Beispiel(Monoton-Monom$_n$), Beispiel(Monom$_n$) $= O(\frac{1}{\varepsilon} \cdot n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

b) Beispiel(k-CNF$_n$), Beispiel(k-DNF$_n$) $= O(\frac{n^k}{\varepsilon} + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

c) Beispiel(k-Klauseln CNF$_n$), Beispiel(k-Term DNF$_n$) $= O(\frac{1}{\varepsilon}k \cdot n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

d) Beispiel(CNF$_n$), Beispiel(DNF$_n$), Beispiel(Boolean$_n$) $= O(\frac{1}{\varepsilon}2^n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

e) Beispiel(Threshold$_n$) $= O(\frac{1}{\varepsilon}n^2 + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

f) Beispiel(Sym$_n$) $= O(\frac{1}{\varepsilon}n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

g) Beispiel(Aut$_{n,\Sigma}$) $= O(\frac{1}{\varepsilon}n \cdot |\Sigma| \cdot \log_2 n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

h) Beispiel(Rechteck$_{n,d}$) $= O(\frac{1}{\varepsilon}n \cdot d + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

Beweis a), b), c), d), f) und g) folgen direkt aus Satz 2.2.1, indem wir $|\mathcal{C}|$ wie folgt berechnen:

a) $|Monoton - Monom_n| = 2^n$, $|Monom_n| = 3^n + 1$,

b) $|k\text{-}CNF_n| = |k\text{-}DNF_n| = O(\sum_{j=0}^{k} \binom{2n}{j}) = 0(n^k)$, falls k konstant,

c) $|k$-Klauseln $CNF_n| = |k-$Term $DNF_n| \leq 2^{2n \cdot k}$,

d) $|CNF_n| = |DNF_n| = |Boolean_n| = 2^{2^n}$,

f) $|Sym_n| = 2^{n+1}$,

g) $|Aut_{n,\Sigma}| \leq n \cdot n^{|\Sigma| \cdot n}$.

e) und h) folgen aus der Bestimmung der VC-Dimension (Satz 2.1.3) und Satz 2.2.1. □

Bemerkung Alle Ergebnisse aus Satz 2.2.2 bis auf Aussage e) und Aussage h) sind optimal. Wir führen im nächsten Kapitel einen Optimalitätsbeweis.

2.3 Eine untere Schranke für die Beispielsanzahl

Unser Ziel ist der Beweis von

2.3.1 Satz *Für jede Konzeptklasse $\mathcal{C}$ gilt*

$$Beispiel(\mathcal{C}) = \Omega\left(\frac{1}{\varepsilon}\left(VC(\mathcal{C}) + \ln(\frac{1}{\delta})\right)\right),$$

falls $VC(\mathcal{C}) \geq 3$, $\delta \leq \frac{1}{70}$ und $\varepsilon \leq \frac{1}{32}$ ist.

Wir werden nur einen partiellen Beweis geben, und zwar den Beweis von

$$Beispiel(\mathcal{C}) = \Omega\left(\frac{VC(\mathcal{C})}{\varepsilon}\right).$$

Für den vollständigen Beweis von Satz 2.3.1 verweisen wir auf: A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth, *Learnability and the Vapnik-Chervonenkis dimension*, Journal of the ACM, 36 (4),pp. 929-965, 1989.

Beweis von $Beispiel(\mathcal{C}) = \Omega(\frac{VC(\mathcal{C})}{\varepsilon})$:
Da wir die Behauptung für jede Verteilung zeigen müssen, konstruieren wir zuerst eine schwierige Verteilung D. Dazu nehmen wir an, daß die Konzeptklasse $\mathcal{C}$, mit $VC(\mathcal{C}) \geq 3$, die Menge $S = \{x_+, x_-, x_1, \ldots, x_d\}$ zertrümmert, wobei $d = VC(\mathcal{C}) - 2$ ist. Wir wählen die Verteilung D so, daß nur Beispiele aus der Menge S generiert werden; dabei sollen die Beispiele x_+ und x_- „hochwahrscheinlich“ und die restlichen Elemente aus S alle gleich „wenig wahrscheinlich“ sein:

$$\begin{array}{lllll} \text{Für} & x \notin S & \text{setze} & D(x) & = & 0, \\ & & \text{setze} & D(x_+) & = & \frac{1}{2} - 16\varepsilon, \\ & & \text{setze} & D(x_-) & = & \frac{1}{2} - 16\varepsilon, \\ \text{für} & 1 \leq i \leq d & \text{setze} & D(x_i) & = & \frac{32\varepsilon}{d}. \end{array}$$

Wir werden von jetzt an nur noch die Konzeptklasse

$$\mathcal{C}' = \{c \cap S \mid c \in \mathcal{C}, x_+ \in c, x_- \notin c\}$$

betrachten. (Dies ist möglich, da die Verteilung D allen Beispielen aus $\bar{S}$ die Wahrscheinlichkeit 0 gibt, und damit diese Beispiele weder beim Lernen noch beim Testen auftauchen. Wir betrachten $\mathcal{C}'$ also als eine Konzeptklasse über S.) Beachte, daß $\mathcal{C}'$ die Menge $\{x_1, \ldots, x_d\}$ zertrümmert.

Wir führen den Beweis in 2 Schritten durch. **Zuerst** beweisen wir, daß eine Beispielmenge mit relativ hoher Wahrscheinlichkeit weniger als $\frac{d}{2}$ Beispiele aus der Menge $\{x_1, \ldots, x_d\}$

besitzt, falls höchstens $s \leq \frac{d}{128\varepsilon}$ Beispiele angefordert werden. Dies ist nicht allzu überraschend, da die Elemente x_+ und x_- sehr häufig gewählt werden. Mit anderen Worten: Falls die Anzahl der Beispiele zu klein ist, erhalten wir nicht genügend Elemente aus $\{x_1, \ldots, x_d\}$, um einen guten Überblick über S zu gewinnen und können das Konzept nicht herausfinden.

Wir nennen eine Beispielsmenge, die weniger als $\frac{d}{2}$ Beispiele aus $\{x_1, \ldots, x_d\}$ enthält, schlecht. **Im zweiten Schritt** nutzen wir aus, daß C' die Menge $\{x_1, \ldots, x_d\}$ zertrümmert und beweisen, daß bei einer schlechten Beispielsmenge der Fehler (mit relativ großer Wahrscheinlichkeit) zu groß wird.

Für den **ersten Schritt** beweisen wir das folgende Lemma:

2.3.2 Lemma Sei $c \in C'$. Sei $T \subseteq S$ eine, gemäß D, zufällig gewählte, mit c konsistente Beispielmenge von s Beispielen. Wenn $s \leq \frac{d}{128\varepsilon}$ ist, dann gilt:

$$prob[T \text{ ist schlecht}] \geq 7\delta.$$

Beweis Beachte, daß die Anzahl der angeforderten Beispielen und die Anzahl der **verschiedenen** Beispielen in der erhaltenen Beispielmenge unterschiedlich ist, d.h. T ist eine Teilmenge von S.
Wir wollen zunächst die Anzahl der aus $\{x_1, \ldots, x_d\}$ generierten Beispiele abschätzen. Dazu definieren wir Zufallsvariablen $Y_1, \ldots, Y_s$ mit

$$Y_i = \begin{cases} 1 & \text{das i} - \text{te Beispiel gehört zu } \{x_1, \ldots, x_d\} \\ 0 & \text{sonst.} \end{cases}$$

Dann ist $\sum_{i=1}^{s} Y_i \geq$ Anzahl der aus $\{x_1, \ldots, x_d\}$ generierten Beispiele.
Mit welcher Wahrscheinlichkeit wird ein Beispiel aus der Menge $\{x_1, \ldots, x_d\}$ gewählt?

$$prob[Y_i = 1] = 1 - \underbrace{(1 - 32\varepsilon)}_{\text{Wahrscheinlichkeit von } x_+ \text{ und } x_-} = 32\varepsilon.$$

Da wir s unabhängige Versuche haben, ist $32\varepsilon \cdot s$ der Erwartungswert von $\sum_{i=1}^{s} Y_i$.

Es gilt

$$\begin{aligned} prob[T \text{ ist } \textbf{nicht} \text{ schlecht}] &\leq prob[\text{mehr als } \tfrac{d}{2} \text{ mal wird ein Beispiel aus} \\ &\qquad \{x_1, \ldots, x_d\} \text{ gewählt}] \\ &= prob[\sum_{i=1}^{s} Y_i \geq \tfrac{d}{2}]. \end{aligned}$$

Je mehr Beispiele angefordert werden, desto höher ist die Wahrscheinlichkeit, daß die Beispielsmenge **nicht** schlecht ist. Diese Wahrscheinlichkeit wächst also mit wachsendem s und ist maximal für $s = \frac{d}{128\varepsilon}$. Der Erwartungswert von $\sum_{i=1}^{s} Y_i$ ist dann $\frac{32 \cdot \varepsilon \cdot d}{128 \cdot \varepsilon} = \frac{d}{4}$.

Wir sind mit $\frac{d}{2}$ somit zweifach über dem Erwartungswert, dies ist nach Chernoff sehr unwahrscheinlich:

$$prob[\sum_{i=1}^{s} Y_i \geq (1+\alpha) \cdot \frac{d}{4}] \leq e^{\frac{-\alpha^2 \frac{d}{4}}{3}},$$

für $0 \leq \alpha \leq 1$. Für $\alpha = 1$ ist $(1+\alpha) \cdot \frac{d}{4} = \frac{d}{2}$ und wir erhalten:

$$\begin{aligned} prob[T \text{ ist } \textbf{nicht} \text{ schlecht}] &\leq e^{-\frac{d}{12}} \\ &\leq e^{-\frac{1}{4}}, \quad \text{da } d \geq 3. \end{aligned}$$

Somit gilt für die Wahrscheinlichkeit, daß die Beispielsmenge T schlecht ist:

$$prob[T \text{ ist schlecht}] \geq 1 - \mathrm{e}^{-\frac{1}{4}} \geq 7\delta,$$

falls $\delta < \frac{1}{70}$ ist.
(Die letzte Ungleichung weisen hier wir nicht nach.) □

Für den **zweiten Schritt** des Beweises von Satz 2.3.1 zeigen wir das folgende Lemma:

2.3.3 Lemma Es gibt ein Konzept $c_0 \in \mathcal{C}'$, so daß

$$prob[Fehler_D(c_0, h) \geq 2\varepsilon \mid \text{die Beispielsmenge ist schlecht}] \geq \frac{1}{5},$$

wobei h die vom Lernalgorithmus ausgegebene Hypothese ist.

(Mit anderen Worten: Unter der Voraussetzung, daß die erhaltene Beispielsmenge schlecht ist, ist die Wahrscheinlichkeit, daß der Fehler zu groß ist relativ hoch.)

Beweis Beachte, daß wir zwar (aufgrund der bösartigen Verteilung D) mit hoher Wahrscheinlichkeit höchstens die Hälfte aller Elemente aus S sehen, dafür aber auch auf dieser Verteilung D ausgewertet werden.

Für eine **fest vorgegebene** Beispielsmenge T sei $h(T)$ die vom Lernalgorithmus ausgegebene Hypothese. (Da die Beispielsmenge fixiert ist, ist auch die ausgegebene Hypothese fixiert.) Wir betrachten zuerst den **erwarteten** Fehler bei festem T:

$$E_T(Fehler_D(c, h(T))),$$

wobei der Erwartungswert hierbei über alle Konzepte c berechnet wird, die mit T konsistent sind.

Sei $Anzahl(c, h(T))$ die Anzahl der Beispiele aus $\{x_1, \ldots, x_d\}$, in denen sich c und $h(T)$ unterscheiden. Dann erhalten wir

$$Fehler_D(c, h(T)) = \sum_{x \in c \oplus h(T)} D(x) = \frac{32\varepsilon}{d} \cdot Anzahl(c, h(T))$$

und deshalb
$$\begin{aligned} E_T(Fehler_D(c,h(T))) &= E(\tfrac{32\varepsilon}{d} \cdot Anzahl(c,h(T))) \\ &= \tfrac{32\varepsilon}{d} E(Anzahl(c,h(T))). \end{aligned}$$

Auf der Beispielsmenge T ist die Hypothese $h(T)$ mit dem zu lernenden Konzept c konsistent. Wir wissen, daß T eine schlechte Beispielsmenge ist und somit weniger als $\frac{d}{2}$ Beispiele aus $\{x_1,\ldots,x_d\}$ enthält. Außerdem zertrümmert $\mathcal{C}'$ die Menge $\{x_1,\ldots,x_d\}$: Es gibt also ein Konzept für jede Teilmenge von $\{x_1,\ldots,x_d\}$. Das bedeutet aber, daß sich c und $h(T)$ mit Wahrscheinlichkeit $\frac{1}{2}$ auf jedem nicht-erschienenen Beispiel unterscheiden. Somit gilt:
$$E(Anzahl(c,h(T))) \geq \frac{d}{4}.$$

Also erhalten wir:
$$E_T(Fehler_D(c,h(T))) \geq \frac{32\varepsilon}{d} \cdot \frac{d}{4} = 8\varepsilon.$$

Bisher haben wir gezeigt, daß bei *fixierter* schlechter Beispielmenge der erwartete Fehler (berechnet über alle konsistenten Konzepte) groß ist. Im nächsten Schritt suchen wir nach einem Konzept, so daß der erwartete Fehler (berechnet über alle konsistenten, schlechten Beispielsmengen) groß ist.

Betrachte die Matrix, deren Zeilen mit allen schlechten Beispielsmengen und deren Spalten mit den Konzepten aus $\mathcal{C}'$ indiziert sind. In Zeile T und Spalte c trage $Fehler_D(c,h(S))$ ein, falls c mit T konsistent ist. Falls c **nicht** mit T konsistent ist, trage 0 ein.

Für jede Zeile der Matrix ist der durchschnittliche Wert (von Einträgen, die von 0 verschieden sind) mindestens 8ε. Damit ist aber auch der durchschnittliche Wert (von Einträgen, die von 0 verschieden sind) der **gesamten Matrix** mindestens 8ε. Das bedeutet aber, daß es eine Spalte c_0 geben muß, deren durchschnittlicher Wert (für von Null verschiedene Einträge) mindestens 8ε beträgt. (Denn sonst ist der durchschnittliche Wert (der von 0 verschiedenen Einträge) für alle Spalten kleiner als 8ε und somit auch der durchschnittliche Wert (der von 0 verschiedenen Einträge) der gesamten Matrix kleiner als 8ε.)

Wir gehen jetzt zurück zum Erwartungswert, wobei diesmal das Konzept c_0 fest ist, und wir den Erwartungswert über alle mit c_0 konsistenten, schlechten Beispielsmengen den Erwartungswert bilden. Sei also $E_{c_0}(Fehler_D(c_0,h(T)))$ der erwartete Fehler, wenn eine mit c_0 konsistente, schlechte Beispielsmenge T zufällig gewählt wird. Unsere obige Argumentation hat ergeben, daß
$$E_{c_0}(Fehler_D(c_0,h(T))) \geq 8\varepsilon$$
ist. Nun kann es noch passieren, daß es einige wenige mit c_0 konsistente, schlechte Beispielsmengen gibt, auf denen der Fehler enorm groß ist. Dementsprechend ist es möglich, daß der Fehler auf den meisten mit c_0 konsistenten, schlechten Beispielsmengen klein genug ist. Warum kann dies **nicht** passieren?

Schlimmstensfalls beträgt der Fehler 32ε, nämlich dann wenn kein Beispiel aus der Menge $\{x_1, \ldots, x_d\}$ gewählt wird, und c_0 und $h(T)$ sich in allen Beispielen aus $\{x_1, \ldots, x_d\}$ unterscheiden.

Sei $p := \ prob[Fehler_D(c_0, h(T)) \geq 2\varepsilon]$. Wir benutzen, daß für jede Zufallsvariable X die Abschätzung

$$E(X) \leq prob[X \geq T] \cdot \text{maximaler Wert von } X + prob[x < T] \cdot T$$

gilt. Damit ist:

$$\begin{array}{rrcl} & 8\varepsilon & \leq & E_{c_0}(Fehler_D(c_0, h(T))) \leq p \cdot 32\varepsilon + (1-p) \cdot 2\varepsilon \\ \Rightarrow & 8\varepsilon & \leq & 32\varepsilon p + (1-p)2\varepsilon \\ \Rightarrow & 6\varepsilon & \leq & 30\varepsilon p \\ \Rightarrow & \frac{1}{5} & \leq & p. \end{array}$$

□

Wir können jetzt den Beweis von Satz 2.3.1 zu Ende führen: Mit Wahrscheinlichkeit mindestens 7δ erhalten wir eine schlechte Beispielsmenge. Mit Wahrscheinlichkeit mindestens $\frac{1}{5}$ wird dann eine Hypothese $h(T)$ gewählt mit $Fehler_D(c_0), h(T))) \geq 2\varepsilon$. Insgesamt gilt also:

$$prob[Fehler_D(c_0, h(T)) \geq 2\varepsilon] \geq 7\delta \cdot \frac{1}{5} > \delta$$

im WIDERSPRUCH zur Forderung

$$prob[Fehler_D(c_0, h(T)) > \varepsilon] < \delta.$$

Somit ist der Lernalgorithmus **kein** PAC-Algorithmus, wenn $s \leq \frac{d}{128\varepsilon} = \frac{VC(\mathcal{C})-2}{128\varepsilon}$. ■

2.3.4 Korollar Für jedes n sei $\mathcal{C}_n$ eine Konzeptklasse über $\Sigma^{\leq n}$ und es gelte

$$\ln(|\mathcal{C}_n|) = O(VC(\mathcal{C}_n)).$$

Dann erhalten wir asymptotisch eine **exakte** untere und obere Schranke für die Anzahl der notwendigen Beispiele:

$$Beispiel(\mathcal{C}_n) = \Theta(\frac{1}{\varepsilon}(VC(\mathcal{C}_n) + \ln(\frac{1}{\delta}))).$$

Beweis Es gilt stets $VC(\mathcal{C}_n) \leq \log_2(|\mathcal{C}_n|)$. Da $\ln(|\mathcal{C}_n|) = O(VC(\mathcal{C}_n))$ erhalten wir

$$\ln(|\mathcal{C}_n|) = \Theta(VC(\mathcal{C}_n)).$$

Die Behauptung folgt nun mit Satz 2.3.1 und Satz 2.2.1.

Bemerkung Wir haben in Satz 2.2.2 also die notwendige Beispielsanzahl asymptotisch exakt bestimmt für die Konzeptklassen

- $Beispiel(Monoton\text{-}Monom_n)$, $Beispiel(Monom_n) = \Theta(\frac{1}{\varepsilon} \cdot n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$,
- $Beispiel(k\text{-}CNF_n)$, $Beispiel(k\text{-}DNF_n) = \Theta(\frac{n^k}{\varepsilon} + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$, falls k konstant ist,
- $Beispiel(k\text{-Klauseln } CNF_n)$, $Beispiel(k\text{-Term } DNF_n) = \Theta(\frac{1}{\varepsilon}k \cdot n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$, falls k konstant ist,
- $Beispiel(CNF_n)$, $Beispiel(DNF_n)$, $Beispiel(Boolean_n) = \Theta(\frac{1}{\varepsilon}2^n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$,
- $Beispiel(Sym_n) = \Theta(\frac{1}{\varepsilon}n + \frac{1}{\varepsilon}\ln(\frac{1}{\delta}))$.

Andererseits überschätzt Satz 2.2.2 die notwendige Beispielsanzahl für $Threshold_n$ und $Rechteck_{n,d}$. Deshalb leiten wir jetzt eine weitere obere Schranke her.

2.4 Eine verschärfte obere Schranke für die Beispielsanzahl

Satz 2.2.1 ist genau, falls $VC(\mathcal{C}) \approx \ln(|\mathcal{C}|)$ ist. (Wenn $\mathcal{C}$ schwierig ist, dann wird $VC(\mathcal{C})$ groß sein.) Die obere Schranke für die Beispielsanzahl aus Satz 2.2.1 fordert jedoch zu viele Beispiele an, falls die VC-Dimension wesentlich unter dem Logarithmus der Anzahl der Konzepte liegt (wenn also $VC(\mathcal{C}) \ll \ln(|\mathcal{C}|)$ ist). In diesem Fall werden wir eine obere Schranke angeben, die wesentlich von der VC-Dimension der Konzeptklasse abhängt.

Wir benötigen das folgende technische Lemma.

2.4.1 Lemma Für ein Konzept $c \subseteq \Sigma^*$ sei $\hat{c} = \{w1 \mid w \in c\} \cup \{w0 \mid w \notin c\}$. Dann gilt

$$VC(\mathcal{C}) = VC(\hat{\mathcal{C}})$$

für jede Konzeptklasse $\mathcal{C}$, wobei $\hat{\mathcal{C}} = \{\hat{c} \mid c \in \mathcal{C}\}$ ist.

Beweis Wir zeigen zuerst $VC(\hat{\mathcal{C}}) \geq VC(\mathcal{C})$. Sei $S \subset \Sigma^*$ eine Menge, die von $\mathcal{C}$ zertrümmert wird. Es genügt zu zeigen, daß $\hat{\mathcal{C}}$ die Menge

$$S' = \{s1 | s \in S\}$$

zertrümmert. Dies ist aber offensichtlich.
Für die Umkehrung (also $(VC(\hat{\mathcal{C}}) \leq VC(\mathcal{C}))$ nehmen wir an, daß $\hat{\mathcal{C}}$ eine Menge $\hat{S} \subseteq \sum^*$ zertrümmert. Es genügt zu zeigen, daß $\mathcal{C}$ die Menge

$$S = \{s \mid s0 \in \hat{S} \text{ oder } s1 \in \hat{S}\}$$

zertrümmert. (Beachte, daß nicht gleichzeitig $s0 \in \hat{S}$ und $s1 \in \hat{S}$ gelten kann: kein Konzept in $\hat{\mathcal{C}}$ enthält $s0$ **und** $s1$. Wir erhalten also $|S| = |\hat{S}|$.)

Sei T eine beliebige Teilmenge von S. Wir setzen

$$\hat{T} = \{x0 \mid x0 \in \hat{S}, x \notin T\} \ \cup \ \{x1 \mid x1 \in \hat{S}, x \in T\}.$$

Nach Voraussetzung gibt es $\hat{c} \in \hat{\mathcal{C}}$ mit $\hat{S} \cap \hat{c} = \hat{T}$. Wir behaupten, daß $S \cap c = T$ ist.

Es gilt $S \cap c \subseteq T$: Denn für $s \in S \cap c$ gilt insbesondere $s \in c$ und damit $s1 \in \hat{c}$. Wenn $s0 \in \hat{S}$, dann $s0 \notin \hat{T}$ (da $\hat{S} \cap \hat{c} = \hat{T}$). Das aber bedeutet nach Definition von $\hat{T}$, daß $s \in T$. Wenn andererseits $s1 \in \hat{S}$, dann ist $s1 \in \hat{S} \cap \hat{c} = \hat{T}$ und nach Definition von $\hat{T}$ folgt $s \in T$.

Es gilt $S \cap c \supseteq T$: Denn für $s \in T$ ist entweder $s1 \in \hat{T}$ oder $s1 \notin \hat{T}$. Im ersten Fall ($s1 \in \hat{T}$) muß $s1 \in \hat{S} \cap \hat{c}$ gelten, da $\hat{S} \cap \hat{c} = \hat{T}$; es folgt, daß $s \in S \cap c$. Im zweiten Fall ($s1 \notin \hat{T}$) ist, nach Definition von $\hat{T}$, $s1 \notin \hat{S}$ und damit $s0 \in \hat{S}$. Es muß somit $s1 \in \hat{c}$ gelten, da ansonsten $s0 \in \hat{c}$ ist, und wir mit $s0 \in \hat{S} \cap \hat{c} = \hat{T}$ einen Widerspruch erhalten. Es folgt also auch hier, daß $s \in S \cap c$. □

Wir können jetzt unsere verbesserte obere Schranke angeben:

2.4.2 Satz *Sei $\mathcal{C}$ eine Konzeptklasse mit $VC(\mathcal{C}) = d$. Dann gibt es einen PAC-Lernalgorithmus für $\mathcal{C}$, der höchstens*

$$s_d(\varepsilon, \delta) = \frac{32}{\varepsilon}\left(4 \cdot d \cdot \ln\left(\frac{6}{\varepsilon}\right) + ln\left(\frac{2}{\delta}\right)\right)$$

Beispiele anfordert.

Beweis Der Lernalgorithmus fordert $s = s_d(\varepsilon, \delta)$ Beispiele an und konstruiert eine beliebige, konsistente Hypothese. Wir müssen zeigen, daß

$$p := prob[Fehler_D(c, h_c) > \varepsilon] \leq \delta$$

gilt. (Das Konzept c ist bei unbekannter Verteilung D zu erlernen. Die Hypothese h_c ist konsistent mit der gemäß D produzierten Beispielsmenge.)

Da wir nicht wissen, welche konsistente Hypothese gewählt wird, nehmen wir an, daß der Algorithmus eine schlechte konsistente Hypothese ausgibt. Für

$$q := prob[\text{Es gibt eine konsistente Hypothese } h \in \mathcal{C} \text{ mit } Fehler_D(c, h) > \varepsilon]$$

gilt offensichtlich $p \leq q$. Um q genauer zu untersuchen, definieren wir

$$E = \{x \mid \begin{array}{l} x \text{ ist ein Vektor von } s \text{ gemäß } c \text{ klassifizierten Beispielen und} \\ \text{es gibt eine mit } x \text{ konsistente Hypothese } h \in \mathcal{C} \text{ mit } Fehler_D(c, h) > \varepsilon\} \end{array}.$$

Dann ist $q = prob[E]$.

Im **ersten Beweisschritt** betrachten wir die Menge

$$J = \{(x, y) \mid \begin{array}{l} x \text{ und } y \text{ sind Vektoren von je } s \text{ gemäß } c \text{ klassifizierten Beispielen, und} \\ \text{es gibt eine mit } x \text{ konsistente Hypothese } h \in \mathcal{C} \text{ mit } Fehler_D(c, h) > \varepsilon, \\ \text{und } h \text{ widerspricht mindestens } \frac{\varepsilon \cdot s}{2} \text{ Beispielen aus } y\}. \end{array}$$

Wir zeigen in Lemma 2.4.3, daß $prob[E] \leq 2 \cdot prob[J]$ gilt. (Dies ist alles andere als überraschend, da eine Hypothese, die einen großen Fehler hat, mit hoher Wahrscheinlichkeit Fehler auf den y-Beispielen macht: h_x, mit $Fehler_D(c, h_x) > \varepsilon$, besitzt eine erwartete Anzahl von mindestens $\varepsilon \cdot s$ falsch klassifizierten Beispielen in y.)

Wenn wir in der Definition von J die Ungleichung „$Fehler_D(c, h_x) > \varepsilon$" weglassen, erhalten wir die größere Menge K und somit $prob[J] \leq prob[K]$. Im **zweiten Beweisschritt** zeigen wir in Lemma 2.4.4, daß

$$prob[K] \leq \Phi(2s, d) \cdot 2^{-\frac{\varepsilon \cdot s}{2}}$$

gilt, wobei $d = VC(\mathcal{C})$ und $\Phi(2s, d) = \sum_{i=0}^{d} \binom{2s}{i}$ ist. Insgesamt erhalten wir dann also, daß

$$prob[Fehler_D(c, h_c) > \varepsilon] \leq 2 \cdot \Phi(2c, d) \cdot 2^{-\frac{\varepsilon \cdot s}{2}}$$

gilt. Wir müssen also noch zeigen, daß die rechte Seite durch δ nach oben beschränkt werden kann.

Im Beweis von Lemma 2.1.5 haben wir nachgewiesen, daß

$$\Phi(2s, d) \leq \left(\frac{e \cdot 2s}{d}\right)^d$$

gilt. Also bleibt zu zeigen, daß

$$2 \cdot \left(\frac{e \cdot 2s}{d}\right)^d \cdot 2^{-\frac{\varepsilon \cdot s}{2}} \leq \delta.$$

Nach Logarithmierung folgt

$$d \cdot \ln(\frac{e \cdot 2s}{d}) - \frac{\varepsilon s}{2} \ln(2) \leq \ln\left(\frac{\delta}{2}\right)$$

und äquivalent

$$d \cdot \ln(\frac{e \cdot 2s}{d}) - \frac{\varepsilon s}{4} \ln(2) \leq \frac{\varepsilon s}{4} \ln(2) - \ln\left(\frac{2}{\delta}\right).$$

Behauptung Es gilt: $\frac{\varepsilon \cdot s}{4} ln(2) \geq ln\left(\frac{2}{\delta}\right)$.

Beweis Wir wissen, nach Definition von s, daß

$$s \geq \frac{32}{\varepsilon} \ln\left(\frac{2}{\delta}\right)$$

ist. In der Behauptung wird gefordert, daß

$$s \geq \frac{4}{\varepsilon \cdot \ln(2)} \ln(\frac{2}{\delta})$$

ist. Diese Ungleichung folgt, da $2 \geq \frac{1}{\ln(2)}$ ist. (Warum? Es gilt: $2\ln(2) = \ln(4) > 1$). □

Behauptung Es gilt: $\frac{\varepsilon \cdot s}{4}\ln(2) \geq d \cdot \ln(\frac{e \cdot 2s}{d})$.

Beweis Die Ungleichung ist äquivalent mit

$$\begin{aligned} \varepsilon \cdot \tfrac{s}{d} &\geq \tfrac{4}{\ln(2)} \cdot \ln\left(\varepsilon \cdot \tfrac{s}{d} \cdot \tfrac{2e}{\varepsilon}\right) \\ &= \tfrac{4}{\ln(2)} \cdot \ln\left(\tfrac{\varepsilon \cdot s}{d}\right) + \tfrac{4}{\ln(2)} \cdot \ln\left(\tfrac{2e}{\varepsilon}\right). \end{aligned}$$

Nach Definition von s ist

$$\begin{aligned} s &\geq 128\tfrac{d}{\varepsilon} \cdot \ln\left(\tfrac{6}{\varepsilon}\right) \\ \Leftrightarrow\ \varepsilon\left(\tfrac{s}{d}\right) &\geq 128 \cdot \ln\left(\tfrac{6}{\varepsilon}\right) \\ \Leftrightarrow\ \tfrac{\varepsilon}{2} \cdot \tfrac{s}{d} &\geq 64\ln\left(\tfrac{6}{\varepsilon}\right) \\ &\geq \tfrac{4}{\ln(2)} \cdot \ln(\tfrac{2e}{\varepsilon}), \end{aligned}$$

denn $\frac{4}{\ln(2)} \leq 8$ und $\frac{2e}{\varepsilon} \leq \frac{6}{\varepsilon}$.
Es bleibt zu zeigen, daß

$$\frac{\varepsilon}{2} \cdot \frac{s}{d} \geq \frac{4}{\ln(2)} \ln\left(\frac{\varepsilon \cdot s}{d}\right)$$

Da $\frac{4}{\ln(2)} \leq 8$, genügt der Nachweis von

$$\varepsilon\frac{s}{d} \geq 16\ln\left(\varepsilon\frac{s}{d}\right).$$

Beachte, daß $t \geq 16\ln(t)$ für $t \geq 128$ erfüllt ist (denn $128 \geq 16 \cdot \log_2 128 \geq 16 \cdot \ln(128)$). Aber $\varepsilon \cdot \frac{s}{d} \geq 128 \cdot \ln(\frac{6}{\varepsilon}) \geq 128$ nach Definition von s und dies war zu zeigen. □

Wir müssen jetzt noch die beiden Lemmata nachweisen. Zuerst:

2.4.3 Lemma Es gilt $prob[E] \leq 2 \cdot prob[J]$.

Beweis Es ist $prob[J] = \sum_{(x,y)\in J} D(x) \cdot D(y)$.

D ist hierbei die unbekannte Verteilung, die wir auf **klassifizierte** Beispiele fortgesetzt haben: $D(\xi, b) = D(\xi)$ für ein Beispiel ξ und Klassifizierung b.

In der obigen Summe ist x (wie auch y) ein Vektor von s klassifizierten Beispielen. Für $x = (x_1, \ldots, x_s)$ ($y = (y_1, \ldots, y_s)$) setzen wir $D(x) = D(x_1) \cdot \ldots \cdot D(x_s)$ ($D(y) = D(y_1) \cdot \ldots \cdot D(y_s)$). Für jeden Vektor x wählen wir jetzt eine Hypothese h_x, so daß h_x konsistent mit x ist, aber einen großen Fehler besitzt (d.h. $Fehler_D(c, h_x) > \varepsilon$). Beachte, daß ein solches h_x nur für $x \in E$ existiert. Wir setzen

$$J' = \{(x,y) \mid \begin{array}{l} x \text{ und } y \text{ sind Vektoren von je } s \text{ klassifizierten Beispielen mit } x \in E, \\ \text{und es gibt mindestens } \frac{\varepsilon \cdot s}{2} \text{ Beispiele in } y, \\ \text{die von } h_x \text{ falsch klassifiziert werden}\}. \end{array}$$

Dann ist $J' \subseteq J$ und deshalb

$$\begin{aligned} prob[J] \geq prob[J'] &= \sum_{(x,y) \in J'} D(x) \cdot D(y) \\ &= \sum_{x \in E} D(x) \cdot \sum_{\substack{y \\ (x,y) \in J'}} D(y). \end{aligned}$$

Beachte, daß

$$\sum_{\substack{y \\ (x,y) \in J'}} D(y) = prob\ [h_x \text{ widerspricht mindestens } \frac{\varepsilon \cdot s}{2} \text{ Beispielen in y}].$$

Das Lemma folgt, wenn wir gezeigt haben, daß

$$\frac{1}{2} \leq prob\ [h_x \text{ widerspricht mindestens } \frac{\varepsilon \cdot s}{2} \text{ Beispielen in y}]$$

für jeden Vektor x gilt. (Denn dann erhalten wir $prob[J] \geq \sum_{x \in E} D(x) \cdot \frac{1}{2} = \frac{1}{2} \cdot prob[E]$.)

Sei x fest gewählt. Wenn wir ein klassifiziertes Beispiel (ξ, b) zufällig wählen, dann ist h_x, mit Wahrscheinlichkeit mindestens ε, nicht konsistent mit der Klassifikation b. Dementsprechend definieren wir die Zufallsvariablen Z_i, $1 \leq i \leq s$, durch

$$Z_i = \begin{cases} 1 & h_x \text{ nicht konsistent mit } (y_i, c_i) \\ 0 & \text{sonst.} \end{cases}$$

Damit ist $Z = \sum_{i=1}^{s} Z_i$ eine obere Schranke für die Anzahl der nicht-konsistenten y-Beispiele. Der Erwartungswert von Z sei $E(Z)$ und es gilt

$$E(Z) = E\left(\sum_{i=1}^{s} Z_i\right) = \sum_{i=1}^{s} E(Z_i) \geq \sum_{i=1}^{s}(1 \cdot \varepsilon + 0 \cdot (1-\varepsilon)) = s \cdot \varepsilon$$

(denn die Wahrscheinlichkeit für Nichtkonsistenz ist $prob[Z_i = 1] \geq \varepsilon$). Wir können jetzt die Chernoff-Schranke (mit $\alpha = \frac{1}{2}$) anwenden und erhalten

$$\begin{aligned} prob\left[\sum_{i=1}^{s} Z_i \leq \frac{1}{2} E\left(\sum_{i=1}^{s} Z_i\right)\right] &\leq e^{-\frac{\left(\frac{1}{2}\right)^2 E(\sum_{i=1}^{s} Z_i)}{2}} \\ &\leq e^{-\frac{\left(\frac{1}{2}\right)^2 \varepsilon \cdot s}{2}}. \end{aligned}$$

Gemäß der Behauptung von Satz 2.4.2 ist $s \geq \frac{32}{\varepsilon}(4 \cdot \underbrace{VC(\mathcal{C})}_{>1} \cdot \underbrace{\ln(\frac{6}{\varepsilon})}_{>1}) \geq \frac{128}{\varepsilon}$, d.h. $\varepsilon \cdot s \geq 128$.

Deshalb gilt:

$$\begin{aligned} prob\left[\sum_{i=1}^{s} Z_i \leq \frac{s \cdot \varepsilon}{2}\right] &\leq prob\left[\sum_{i=1}^{s} Z_i \leq \frac{1}{2} E\left(\sum_{i=1}^{s} Z_i\right)\right] \\ &\leq e^{-\frac{\left(\frac{1}{2}\right)^2 \varepsilon \cdot s}{2}} \\ &\leq e^{-\frac{128}{8}} = e^{-16} < \tfrac{1}{2}. \end{aligned}$$

Wir haben also gezeigt, daß

$$\sum_{\substack{y \\ (x,y) \in J'}} D(Y) = prob[h_x \text{ verstößt gegen mindestens } \tfrac{\varepsilon \cdot s}{2} \ y\text{-Beispiele}] \geq \frac{1}{2}$$

ist. □

2.4.4 Lemma Es sei

$K = \{(x,y) \mid$ x und y sind Vektoren von je s gemäß c klassifizierten Beispielen, und es gibt eine mit x konsistente Hypothese $h \in \mathcal{C}$, die mindestens $\frac{\varepsilon \cdot s}{2}$ y-Beispiele mißklassifiziert$\}$.

Dann gilt
a) $J \subseteq K$ und
b) $prob[K] \leq \Phi(2s,d) \cdot 2^{-\frac{\varepsilon \cdot s}{2}}$, mit $\Phi(2s,d) = \sum\limits_{i=0}^{d} \binom{2s}{i} \leq \left(\frac{e \cdot 2s}{d}\right)^d$.

Beweis Teil a) ist klar, da die Menge J noch die zusätzliche Anforderung „Hypothese h ist schlecht“ besitzt.
b) Es ist $prob[K] = \sum\limits_{(x,y) \in K} D(x) \cdot D(y)$.
Für die Fehlerberechnung ist die Reihenfolge der Beispiele nicht wichtig, nur die Menge der erhaltenen Beispiele ist wichtig. Um dies auszudrücken permutieren wir die Folge (x,y) mit Permutationen $\pi \in S_{2s}$. (S_{2s} enthält alle Permutationen von $2s$ Objekten.) Insbesondere gilt für jedes $\pi \in S_{2n}$

$$prob[K] = \sum_{\pi(x,y) \in K} D(x) \cdot D(y)$$

und deshalb

$$\begin{aligned}(2s)!\ prob\ [K] &= \sum_{\pi \in S_{2s}} \sum_{\pi(x,y) \in K} D(x) \cdot D(y) \\ &= \sum_{(x,y)} \sum_{\substack{\pi \in S_{2s} \\ \pi(x,y) \in K}} D(x) \cdot D(y) \\ &= \sum_{(x,y)} D(x) \cdot D(y) \sum_{\substack{\pi \in S_{2s} \\ \pi(x,y) \in K}} 1.\end{aligned}$$

Wir wählen jetzt eine beliebige Folge (x_0, y_0), so daß $\pi(x_0, y_0) \in K$ für zumindest eine Permutation π gilt. Für wieviele Permutationen π ist $\pi(x_0, y_0) \in K$? Wir möchten also

$$anzahl_{x_0,y_0} := |\{\pi \in S_{2s} \mid \pi(x_0, y_0) \in K\}|$$

nach oben abschätzen. (Denn $prob[K] = \frac{1}{(2s)!} \cdot \sum_{(x,y)} D(x) \cdot D(y) \cdot anzahl_{x,y}$.)

Wann ist $\pi(x_0, y_0) \in K$? Wenn es eine Hypothese h gibt, die mit den (nach Permutation) ersten s Beispielen konsistent ist und gegen mindestens $\frac{\varepsilon \cdot s}{2}$ Beispielen unter den (nach Permutation) letzten s Beispielen verstößt.

Mit $menge(x_0, y_0)$ bezeichnen wir die Menge der klassifizierten Beispiele aus x_0, y_0. Wir nennen $menge(x_0, y_0) \backslash \hat{h}$ (zur Notation siehe Lemma 2.4.1) einen Zeugen für $\pi(x_0, y_0) \in K$. (Ein Zeuge für $\pi(x_0, y_0) \in K$ besteht also aus der Menge der bzgl. h mißklassifizierten Beispiele.)

Wir definieren $l(h) :=$ Anzahl der Positionen unter den letzten s Beispielen, gegen die h verstößt. Beachte: Für $\pi(x_0, y_0) \in K$ gilt $l(h) \geq \frac{\varepsilon \cdot s}{2}$.

Frage: Gegeben (x_0, y_0) und h. Für wieviele Permutationen π kann $menge(x_0, y_0) \backslash \hat{h}$ Zeuge sein? (Mit anderen Worten: Wieviele Permutationen besitzen die selben mißklassifizierten Beispiele?)
Die Elemente aus $menge(x_0, y_0) \backslash \hat{h}$ können nur unter den letzten s Beispielen vorkommen. Es gibt also $\binom{s}{l(h)} l(h)!$ Möglichkeiten für die Wahl der $l(h)$Positionen. Letztendlich können die übrigen $2s - l(h)$ Positionen (das sind all die Positionen, deren Beispiele konsistent mit h sind) beliebig permutiert werden.

Antwort: Gegeben h. Dann ist $menge(x_0, y_0) \setminus \hat{h}$ Zeuge für

$$\binom{s}{l(h)} \cdot l(h)! \cdot (2s - l(h))!$$

viele Permutationen.

Die folgende Frage ist zentral und bringt die VC-Dimension in's Spiel.

Frage: Gegeben (x_0, y_0). Wieviele Zeugen gibt es?
Jedes Konzept h liefert einen Zeugen $menge(x_0, y_0) \setminus \hat{h}$. Aber im allgemeinen wird ein

Zeuge von vielen Konzepten generiert; dies nutzen wir aus. Zuerst betrachten wir zwei Konzepte $\hat{h}_1, \hat{h}_2 \in \hat{\mathcal{C}}$ (Notation von Lemma 2.4.1) mit

$$menge(x_0, y_0) \cap \hat{h}_1 = menge(x_0, y_0) \cap \hat{h}_2.$$

Dann unterscheiden sich $\hat{h}_1$ und $\hat{h}_2$ **nicht** auf den falsch klassifizierten Beipielen aus (x_0, y_0) und generieren deshalb denselben Zeugen $menge(x_0, y_0) \setminus \hat{h}_1 = menge(x_0, y_0) \setminus \hat{h}_2$. Wir setzen

$$H := \{\hat{h} \cap menge(x_0, y_0) \mid \hat{h} \in \hat{\mathcal{C}}\}$$

und erhalten somit als **Antwort** auf die letzte Frage:

$$\text{Anzahl der Zeugen } \leq |H|.$$

Es ist $|menge(x_0, y_0)| \leq 2s$ (ein Beispiel kann mehrmals auftauchen, deshalb „$\leq$“). Mit Lemma 2.1.5 gilt:

$$\begin{aligned} |H| &\leq \Phi(|menge(x_0, y_0)|, VC(H)) \\ &\leq \Phi(2s, VC(H)). \end{aligned}$$

Weiterhin ist $VC(H) \leq VC(\hat{\mathcal{C}})$, denn in H haben wir die Konzepte aus $\hat{\mathcal{C}}$ auf Elemente aus $menge(x_0, y_0)$ eingeschränkt. Mit Lemma 2.4.1 erhalten wir $VC(H) \leq VC(\mathcal{C})$ und damit

$$|H| \leq \Phi(2s, VC(\mathcal{C})) = \Phi(2s, d).$$

Wir können jetzt die Bilanz ziehen. Für (x_0, y_0) ist

$$\begin{aligned} \tfrac{1}{(2s)!} \cdot anzahl_{x_0,y_0} &= \tfrac{1}{(2s)!} \cdot |\{\pi \in S_{2s} | \pi(x_0, y_0) \in K\}| \\ &\leq \tfrac{1}{(2s)!} \cdot \underbrace{\Phi(2s, d)}_{\text{Anzahl Zeugen}} \cdot \underbrace{\binom{s}{l(h)} \cdot l(h)! \cdot (2s - l(h))!}_{\text{Anzahl Permutationen bei gegebenem Zeugen}} \\ &= \Phi(2s, d) \cdot \tbinom{s}{l(h)} \cdot \tfrac{l(h)! \cdot (2s - l(h))!}{(2s)!} \\ &= \Phi(2s, d) \cdot \frac{\binom{s}{l(h)}}{\binom{2s}{l(h)}} \\ &\underset{*}{\leq} \Phi(2s, d) \cdot 2^{-l(h)} \\ &\leq \Phi(2s, d) \cdot 2^{-\frac{\varepsilon \cdot s}{2}}, \end{aligned}$$

wobei $*$ gilt, da allgemein $\binom{r}{t} / \binom{2r}{t} = \frac{r}{2r} \cdot \frac{r-1}{2r-1} \cdot \ldots \cdot \frac{r-t+1}{2r-t+1} \leq 2^{-t}$ ist.

Die Behauptung des Lemmas 2.4.4 folgt, denn

$$\begin{aligned}
prob[K] &= \sum_{(x,y)} \frac{1}{(2s)!} \sum_{\substack{\pi\in S_{2s}\\ \pi(x,y)\in K}} D(x)\cdot D(y)\\
&= \sum_{(x,y)} \frac{1}{(2s)!}\cdot anzahl_{x,y}\cdot D(x)\cdot D(y)\\
&\leq \sum_{(x,y)} \Phi(2s,d)\cdot 2^{-\frac{\varepsilon\cdot s}{2}}\cdot D(x)\cdot D(y)\\
&= \Phi(2s,d)\cdot 2^{-\frac{\varepsilon\cdot s}{2}} \underbrace{\sum_{(x,y)} \cdot D(x)\cdot D(y)}_{=(\sum_x D(x))(\sum_y D(y))=1}\\
&= \Phi(2s,d)\cdot 2^{-\frac{\varepsilon\cdot s}{2}}
\end{aligned}$$

wie behauptet. ■

Wir stellen noch einmal heraus, wie die VC-Dimension den Fehler beeinflußt:

2.4.5 Korollar Sei $c \in \mathcal{C}$ und das Konzept h_c sei konsistent mit s (gemäß c klassifizierten) Beispielen. Die Beispiele werden gemäß Verteilung D erzeugt. Dann gilt:

$$prob[Fehler_D(c,h_c) \geq \varepsilon) \leq 2\cdot\Phi(2s, VC(\mathcal{C}))\cdot 2^{-\frac{\varepsilon\cdot s}{2}}.$$

Wir können jetzt die geforderte Anzahl an Beispielen für $Threshold_n$ und $Rechteck_{n,d}$, verglichen mit Satz 2.2.2, wesentlich verbessern (vergleiche Satz 2.3.1 zur unteren Schranke):

2.4.6 Korollar Wir erhalten die folgenden Schranken für die Beispielsanzahl von $Rechteck_{n,d}$ und $Threshold_n$:
a) $\Omega\left(\frac{1}{\varepsilon}\left(d+\ln\left(\frac{1}{\delta}\right)\right)\right) = Beispiel(Rechteck_{n,d}) = O\left(\frac{1}{\varepsilon}\left(d\cdot\ln\left(\frac{1}{\varepsilon}\right)+\ln\left(\frac{1}{\delta}\right)\right)\right)$,
b) $\Omega\left(\frac{1}{\varepsilon}\left(n+\ln\left(\frac{1}{\delta}\right)\right)\right) = Beispiel(Threshold_n) = O\left(\frac{1}{\varepsilon}\left(n\cdot\ln\left(\frac{1}{\varepsilon}\right)+\ln\left(\frac{1}{\delta}\right)\right)\right)$.

Die obere Schranke von Satz 2.4.2 gilt auch für reellwertige Konzeptklassen $\mathcal{C}$ (also Konzeptklassen, deren Konzepte in $\mathbb{R}^d$) enthalten sind), wenn wir fordern, daß nur **meßbare** Konzepte erlaubt sind. (Eine Teilmenge T von $\mathbb{R}^d$ ist genau dann meßbar, wenn

- T ein achsenparalleles Rechteck in $\mathbb{R}^d$ ist,
- $\bar{T}$ meßbar ist oder
- T eine abzählbar unendliche Vereinigung von meßbaren Teilmengen ist.)

Der Beweis von Satz 2.4.2 kann dann direkt auf den reellwertigen Fall übertragen werden, wenn die entsprechenden Summen durch Integrale ersetzt werden. Die Meßbarkeit der Konzepte garantiert, daß die Integrale wohldefiniert sind.
Die Bedingung der Meßbarkeit ist keine einschneidende Einschränkung. Als ein Beispiel erhalten wir die Konzeptklasse $Recheck^*_{n,d}$, die als Konzepte achsenparallele Rechtecke besitzt. Im Unterschied zu $Rechteck_{n,d}$ besteht ein Konzept jetzt aus **allen** Punkten des Rechtecks und nicht nur aus Punkten mit ganzzahligen Koeffizienten.

Auch die VC-Dimension kann für reellwertige Konzepte eingeführt werden, und zwar vollständig analog zum Booleschen Fall. So gilt zum Beispiel $VC(Rechteck^*_{n,d}) = 2d$. Die untere Schranke aus Satz 2.3.1 gilt ebenfalls im reellwertigen Fall.

2.5 Das Lernen von Konzeptklassen durch Hypothesenklassen

Bisher haben wir angenommen, daß die Konzeptklasse der Hypothesenklasse entspricht. Wir heben diese Einschränkung jetzt auf.

Gegeben sei eine Konzeptklasse $\mathcal{C}$ sowie eine Hypothesenklasse $\mathcal{H}$. Wir nehmen an, daß $\mathcal{C} \subseteq \mathcal{H}$ ist und, daß der Lernalgorithmus Hypothesen aus $\mathcal{H}$ ausgibt. Wieviele Beipiele sind in diesem Fall ausreichend?

2.5.1 Satz *Seien eine Koneptklasse $\mathcal{C}$ und eine Hypothesenklasse $\mathcal{H}$ mit $\mathcal{C} \subseteq \mathcal{H}$ gegeben. Jeder Lernalgorithmus, der für eine Beispielsmenge eine konsistente Hypothese aus $\mathcal{H}$ ausgibt, ist ein PAC-Algorithmus, solange mindestens*

$$\begin{array}{rcl} s & \geq & \frac{1}{\varepsilon}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{1}{\delta}\right)\right) \\ \textit{oder} & & \\ s & \geq & \frac{32}{\varepsilon}\left[4 \cdot VC(\mathcal{H}) \ln\left(\frac{6}{\varepsilon}\right) + \ln\left(\frac{2}{\delta}\right)\right] \end{array}$$

Beispiele angefordert werden.

Beweis Der Satz ist eine unmittelbare Konsequenz von Satz 2.2.1 bzw. Satz 2.4.2. □

Somit steigt die Beispielsanzahl an, wenn wir eine kleine Konzeptklasse durch eine größere Hypthesenklasse lernen wollen. Trotzdem kann es sinnvoll sein, eine größere Hypothesenklasse zuzulassen: Nämlich dann, wenn das Konsistenzproblem für $\mathcal{H} = \mathcal{C}$ zu schwierig ist (bei „machbarem“ Konsistenzproblem für $\mathcal{H}$).

Wieviele Beispiele sind im Fall $\mathcal{C} \subseteq \mathcal{H}$ notwendig?
Als untere Schranke kann nur

$$\Omega\left(\frac{1}{\varepsilon}(VC(\mathcal{C}) + ln\left(\frac{1}{\delta}\right)\right)$$

für die Beispielsanzahl gezeigt werden.

Wir betrachten jetzt den allgemeinsten Fall: $\mathcal{H}$ und $\mathcal{C}$ sind unvergleichbar. Dies heißt wir nehmen **nicht** an, daß jedes Konzept aus $\mathcal{C}$ auch als Hypothese existiert (also $\mathcal{C} \not\subseteq \mathcal{H}$). Ein schwieriges Konzept soll hierbei durch eine mächtige Hypothesenklasse approximiert werden. Die Hypothesenklasse darf jedoch nicht zu mächtig sein, da sonst zu viele Beispiele angefordert werden müssen. Dieser Fall ist für die Praxis von größter Bedeutung wie das folgende Beispiel zeigt:
Für jede Person bilden wir ein Konzept und nennen die entsprechende Konzeptklasse $\mathcal{C}$. Für eine Person P besteht das Konzept c_P aus den Bitmustern aller möglichen Schriftproben der Person zusammen mit dem Bitmuster der dazugehörenden Entzifferung. Das Ziel des Lernalgorithmus ist die Musterkennung der Schrift für jede Person: Nach dem Training auf Schriftproben und ihre Entzifferung soll das System im Stande sein, die Schriftproben selbständig zu entziffern. Eine fehlerfreie Entzifferung werden wir im allgemeinen nicht erwarten können, d.h. für die Hypothesenklasse $\mathcal{H}$ (z.B. neuronale Netzwerke) gilt $\mathcal{C} \not\subseteq \mathcal{H}$.

Seien $\mathcal{C}$ und $\mathcal{H}$ sowie eine (unbekannte) Verteilung D gegeben. Um den Fall $\mathcal{C} \not\subseteq \mathcal{H}$ zu untersuchen, wählen wir ein (unbekanntes) Konzept $c \in \mathcal{C}$. Wir nehmen an, daß es eine Hypothese $h \in \mathcal{H}$ gibt, die das Zielkonzept $c \in \mathcal{C}$ „gut“ approximiert, d.h. es gilt

$$Fehler_D(c, h) \leq \varepsilon.$$

Wir möchten die Beispielsanzahl s so bestimmen, daß nach Anforderung von s Beispielen eine „gut-approximierende“ Hypothese h_c mit Wahrscheinlichkeit mindestens $1 - \delta$ gefunden werden kann. „Gut-approximierend“ bedeutet dabei

$$Fehler_D(c, h_c) \leq 4\varepsilon.$$

(Wir wissen also, aufgrund der Annahme, daß es eine gute Hypothese gibt. Wir wissen jedoch nicht, wie wir diese gute Hypothese finden können. Unser Ziel ist es, eine nur unwesentlich schlechtere Hypothese zu finden.)

Unser <u>Lernalgorithmus</u> arbeitet wie folgt:

begin
  lies ε, δ und n;
  bestimme s als Funktion von ε, δ und n;
  for $i := 1$ to n
    lies das i-te Beispiel $x_i \in \Sigma^{\leq n}$ mit Klassifikation $b_i \in \{0, 1\}$;
  suche eine Hypothese $h_c \in \mathcal{H}$, die mit mindestens $(1 - 2\varepsilon)s$ Beispielen
  konsistent ist;
  gib eine Fehlermeldung, wenn eine solche Hypothese nicht existiert;
  ansonsten gib h_c aus
end.

Ziel: Bestimme die Beispielsanzahl s so, daß

$$\begin{aligned}
& prob[\text{Algorithmus findet eine Hypothese } h_c \text{ mit } Fehler_D(c,h_c) \leq 4\varepsilon] && \geq 1-\delta \\
\Leftrightarrow\ & prob[\text{Algorithmus gibt eine Fehlermeldung } \textbf{oder } Fehler_D(c,h_c) \geq 4\varepsilon] && \leq \delta \\
\Leftrightarrow\ & prob[\text{Algorithmus gibt eine Fehlermeldung}] + prob[Fehler_D(c,h_c) \geq 4\varepsilon] && \leq \delta.
\end{aligned}$$

Es reicht somit aus, s so zu bestimmen, daß

$$\begin{aligned}
& prob[\text{Algorithmus gibt eine Fehlermeldung}] && \leq \tfrac{\delta}{2} \\
\textbf{und} \quad & \\
& prob[Fehler_D(c,h_c) \geq 4\varepsilon] && \leq \tfrac{\delta}{2}
\end{aligned}$$

gilt.

Wir bestimmen **im ersten Schritt** s so, daß die erste Ungleichung gilt. Eine Hypothese heißt $\underline{(1-2\varepsilon)s\text{-konsistent}}$, wenn sie mit mindestens $(1-2\varepsilon)s$ Beispielen konsistent ist. Zuerst bestimmen wir die Wahrscheinlichkeit p_1, daß es keine $(1-2\varepsilon)s$-konsistente Hypothese gibt. Dazu betrachten wir die nach Annahme existierende Hypothese h mit

$$Fehler_D(c,h) = \varepsilon.$$

Wir erhalten

$$\begin{aligned}
p_1 &= prob[\text{ Algorithmus gibt Fehlermeldung aus }] \\
&\leq prob[\ h \text{ nicht konsistent mit mindestens } 2\varepsilon s \text{ Beispielen }] =: p_2.
\end{aligned}$$

Setze

$$Y_i = \begin{cases} 1 & \text{das i-te Beispiel ist nicht konsistent mit } h \\ 0 & \text{sonst.} \end{cases}$$

Dann gilt

$$p_2 \leq prob\left[\sum_{i=1}^{s} Y_i \geq 2\varepsilon s\right] =: p_3$$

Beachte, daß $prob[$ ein Beispiel gehört zu $c \oplus h] \leq \varepsilon$ gilt, und deshalb ist der Erwartungswert, daß h nicht konsistent mit der Beispielsmenge ist:

$$E\left(\sum_{i=1}^{s} Y_i\right) \leq \varepsilon \cdot s$$

ist. Wir liegen mit $2\varepsilon \cdot s$ also zweifach über dem Erwartungswert. Die Chernoff-Schranke liefert für $\alpha = 1$ die Abschätzung

$$p_3 \leq e^{-\frac{\varepsilon \cdot s}{3}}.$$

Wir erreichen somit $p_1 \leq \frac{\delta}{2}$, wenn wir

$$s \geq \frac{3}{\varepsilon} \ln\left(\frac{2}{\delta}\right)$$

fordern.

Im zweiten Schritt bestimmen wir s so, daß die zweite Ungleichung, und zwar:

$$p_4 := prob[Fehler_D(c, h_c) \geq 4\varepsilon] \leq \frac{\delta}{2}$$

gilt.

Sei $h_c \in \mathcal{H}$ eine beliebige Hypothese mit $Fehler_D(c, h) \geq 4\varepsilon$. Wir möchten die Wahrscheinlichkeit

$$p_5 := prob[h_c \text{ ist mit höchstens } 2\varepsilon \cdot s \text{ Beispielen nicht konsistent}]$$

bestimmen. Setze

$$Z_i = \begin{cases} 1 & \text{das i} - \text{te Beispiel ist nicht konsistent mit } \mathrm{h_c} \\ 0 & \text{sonst.} \end{cases}$$

Dann gilt

$$p_5 \leq \ \mathrm{prob} \left[\sum_{\mathrm{i}=1}^{\mathrm{s}} \mathrm{Z_i} \leq 2\varepsilon \mathrm{s}.\right]$$

Da $E\left(\sum_{i=1}^{s} Z_i\right) \geq 4\varepsilon s$ gilt, verstößt das Ereignis $\sum_{i=1}^{s} Z_i \leq 2\varepsilon s$ gegen den Erwartungswert und wir können die Chernoff-Schranke mit $\alpha = \frac{1}{2}$ anwenden. Wir erhalten die Abschätzung

$$p_5 \leq e^{-\frac{4\varepsilon \cdot s}{4 \cdot 2}} = e^{-\frac{\varepsilon \cdot s}{2}}$$

Jetzt können wir den zweiten Schritt zu Ende führen:

$$\begin{aligned} p_4 \quad &\leq \sum_{\substack{h_c \in \mathcal{H} \\ Fehler_D(c, h_c) \geq 4\varepsilon}} prob\,[h_c \text{ ist mit } \leq 2\varepsilon\text{s Beispielen nicht konsistent}] \\ &\leq \sum_{h_c \in \mathcal{H}} e^{-\frac{\varepsilon s}{2}} \\ &= |\mathcal{H}| \cdot e^{-\frac{\varepsilon s}{2}}. \end{aligned}$$

Um $p_4 \leq \frac{\delta}{2}$ zu erhalten, folgt nach Logarithmierung:

$$\begin{aligned} \ln(|\mathcal{H}|) - \tfrac{\varepsilon s}{2} \quad &\leq \quad \ln\left(\tfrac{\delta}{2}\right) \\ \Leftrightarrow \quad \tfrac{2}{\varepsilon}\left(\ln(|\mathcal{H}|) + \ln\left(\tfrac{2}{\delta}\right)\right) \quad &\leq \quad s. \end{aligned}$$

Wir können zusammenfassen:

2.5.2 Satz *Seien $\mathcal{C}$ und $\mathcal{H}$, sowie eine Verteilung D gegeben. Sei $c \in \mathcal{C}$. Für eine Hypothese $h \in \mathcal{H}$ gelte $Fehler_D(c, h) \leq \varepsilon$. Der obige Lernalgorithmus findet mit Wahrscheinlichkeit mindestens $1 - \delta$ eine Hypothse $h' \in \mathcal{H}$, so daß*

$$Fehler_D(c, h') \leq 4\varepsilon$$

gilt, wenn mindestens

$$s \geq \frac{3}{\varepsilon}\left(\ln(|\mathcal{H}|) + \ln\left(\frac{2}{\delta}\right)\right)$$

Beispiele angefordert werden.

Auch Satz 2.4.2 kann auf die obige Lernsituation angewendet werden und liefert die ausreichende Beispielsanzahl

$$O\left(\frac{1}{\varepsilon}(VC(\mathcal{H}))\ln\frac{1}{\varepsilon}\right) + \ln\left(\frac{1}{\delta}\right).$$

(Wir sehen an den oberen Schranken für die Beispielsanzahl, daß $|\mathcal{H}|$ nicht zu groß sein darf, da sonst zu viele Beispiele angefordert werden müssen.)

2.6 Zusammenfassung

Wir haben uns mit unteren und oberen Schranken für die Beispielsanzahl beschäftigt. Der zentrale Begriff ist dabei die Vapnik-Chervonenkis Dimension $VC(\mathcal{C})$ für eine Konzeptklasse $\mathcal{C}$.

Falls $\ln(|\mathcal{C}_n|) = \Theta(VC(\mathcal{C}_n))$ für eine Familie $\mathcal{C}_n$ von Konzeptklassen auf $\Sigma^{\leq n}$ gilt, ist die benötigte Beispielsanzahl asymptotisch exakt bestimmbar durch

$$\Theta\left[\frac{1}{\varepsilon}\left(VC(\mathcal{C}_n) + \ln\left(\frac{1}{\delta}\right)\right)\right].$$

Im allgemeinen Fall ist

$$O\left(\frac{1}{\varepsilon}\left(VC\left(\mathcal{C}_n\right)\ln\left(\frac{1}{\varepsilon}\right) + \ln\left(\frac{1}{\delta}\right)\right)\right)$$

eine obere Schranke und

$$\Omega\left(\frac{1}{\varepsilon}\left(VC\left(\mathcal{C}_n\right) + \ln\left(\frac{1}{\delta}\right)\right)\right)$$

eine untere Schranke für die Beispielsanzahl.

Diese unteren und oberen Schranken führen zu einer (asymptotisch) exakten Bestimmung der Beispielsanzahl für viele wichtige Konzeptklassen.

Im nächsten Kapitel wenden wir uns dem letzten Problem beim den Entwurf von PAC-Algorithmen zu: Einer effizienten Lösung des Konsistenzproblems.

3 Effiziente PAC-Algorithmen

Ziel dieses Kapitels ist einerseits die Konstruktion von effizienten Algorithmen für das Konsistenzproblem und andererseits (falls es keinen effizienten Algorithmus dafür gibt) der Beweis von Nicht-Existenz.

Wir leiten erste Aussagen über polynomiell lernbare Konzeptklassen her. Zuerst führen wir den Begriff einer parametrisierten Konzeptklasse ein, um asymptotische Aussagen machen zu können.

3.1 Definition Eine parametrisierte Konzeptklasse $\mathcal{C}$ über Σ ist eine Folge $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ von Konzeptklassen, wobei für jedes n und für jedes $c \in \mathcal{C}_n$ gilt, daß $c \subseteq \Sigma^{\leq n}$ ist.

Beispiele von parametisierten Konzeptklassen sind:

Beispiele

- - $Monom = (Monom_n \mid n \in \mathbb{N})$,
  - $Monoton\text{-}Monom = (Monoton\text{-}Monom_n \mid n \in \mathbb{N})$,
  - k-Term $DNF = (k\text{-Term } DNF_n \mid n \in \mathbb{N})$,
  - $k\text{-}DNF = (k\text{-}DNF_n \mid n \in \mathbb{N})$,
  - k-Klausel $CNF = (k\text{-Klausel } CNF_n \mid n \in \mathbb{N})$,
  - $k\text{-}CNF = (k\text{-}CNF_n \mid n \in \mathbb{N})$,
  - $CNF = (CNF_n \mid n \in \mathbb{N})$.

  Wir verwenden jeweils die Binärdarstellung der zu lernenden Formel als Namen. Für k-Term DNF (bzw. k-Klausel CNF) verwenden wir $k\text{-}CNF$ (bzw. $k\text{-}DNF$) als Hypothesenklasse.

- $Sym = (Sym_n \mid n \in \mathbb{N})$. Der Name der symmetrischen Funktion $s : \{0,1\}^n \to \{0,1\}$ ist durch $b_s \in \{0,1\}^{n+1}$ gegeben; dabei ist $b_s(i)$ der Wert, den s auf Eingaben mit i Einsen annimmt.

- $Threshold = (Threshold_n \mid n \in \mathbb{N})$. Der Name einer Threshold-Funktion $t : \{0,1\}^n \to \{0,1\}$ ist durch die Binärkodierung des Vektors $(w_1, \ldots, w_n, t) \in \mathbb{Z}^{n+1}$ gegeben, wobei $t(x_1, \ldots, x_n) = 1 \Leftrightarrow \sum_{i=1}^{n} w_i x_i \geq t$ gilt.

- $Rechteck_d = (Rechteck_{n,d} \mid n \in \mathbb{N})$. Der Name eines Rechtecks R sei die Binärdarstellung des Vektors $(x_1, y_1, \ldots, x_d, y_d)$, wobei

$$x_i = \min \{\mathrm{z_i} \mid \mathrm{z} = (\mathrm{z_1}, \ldots, \mathrm{z_d}) \in \mathbb{Z}^{\mathrm{d}}, \mathrm{z} \in \mathrm{R}\}$$

$$y_i = \min \{\mathrm{z_i} \mid \mathrm{z} = (\mathrm{z_1}, \ldots, \mathrm{z_d}) \in \mathbb{Z}^{\mathrm{d}}, \mathrm{z} \in \mathrm{R}\}$$

  gilt.

Wir definieren jetzt, was wir unter einem **effizienten** PAC-Algorithmus verstehen. Wichtig für die Effizienz ist neben dem Fehlerparameter ε, dem Vertrauensparameter δ und der Eingabelänge n, die Namensfunktion $Name$: Wir müssen genügend Zeit geben, um zumindest einen Namen aufschreiben zu können.

3.2 Definition Sei $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ eine parametisierte Konzeptklasse mit Namensfunktion $Name : \bigcup_n \mathcal{C}_n \rightarrow \mathcal{P}(\{0,1\}^*)$. Ein Lernalgorithmus L ist genau dann ein polynomieller PAC-Algorithmus für $\mathcal{C}$ und $Name$, wenn für alle ε $(0 < \varepsilon \leq 1)$, für alle δ $(0 < \delta \leq 1)$, für alle $n \in \mathbb{N}$, für alle Verteilungen D auf $\Sigma^{\leq n}$ und für alle Konzepte $c \in \mathcal{C}_n$

$$prob[Fehler_D(c, h_c) \leq \varepsilon] \geq 1 - \delta$$

gilt (wobei h_c die von L ausgegebene Hypothese ist) **und**, wenn L einen Namen aus $Name(h_c)$ in Zeit höchstens

$$poly\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n, \min\{|w| \mid w \in Name(c)\}\right)$$

ausgibt.

3.3 Satz *Die folgenden parametisierten Konzeptklassen besitzen polynomielle PAC-Algorithmen:*

a) Monoton−Monom, Monom, k-Term DNF (mit $\mathcal{H}$ = k-CNF), k-DNF, k-Klausel CNF (mit $\mathcal{H}$ = k-DNF), k-CNF, für konstantes k,

b) Sym,

c) Threshold,

d) Rechteck$_d$.

Beweis a) Jede der erwähnten Konzeptklassen ist entweder in k-CNF oder in k-DNF enthalten. Da k-DNF_n das Komplement von k-CNF_n ist, genügt der Nachweis der polynomiellen Lernbarkeit für k-CNF_n. Es ist ausreichend, eine Lösung des Konsistenzproblems für k-CNF_n zu konstruieren, die in Zeit $poly\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n\right)$ läuft. Ein polynomieller Lernalgorithmus für k-CNF (k konstant):

- Fordere $\Theta\left(\frac{1}{\varepsilon}\left(n^k + \ln\left(\frac{1}{\delta}\right)\right)\right)$ Beispiele an.
- Konstruiere eine mit den Beispielen konsistente k-CNF Formel:
  - Erstelle eine Liste $(k_1, \ldots, k_t)$ aller Klauseln mit höchstens k Literalen.
  - Die anfängliche Hypothese ist $h = \wedge_{i=1}^{t} k_i$, mit $wahr(h) = \emptyset$.

– Entferne für jedes positive Beispiel $x \in \{0,1\}^n$, alle Klauseln k_i, die x widersprechen.

– Die resultierende Hypothese h wird ausgegeben.

Laufzeit: Dieses Verfahren läuft offensichtlich in Zeit $poly\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n\right)$. (Es gibt nur $t = \Theta(n^k)$ viele Klauseln mit höchstens k Literalen.)

Es ist also nur noch zu zeigen, daß h das Konsistenzproblem löst. Dies ist offensichtlich für positive Beispiele. Betrachten wir also ein negatives Beispiel y. Angenommen $h(y) = 1$. Dann gibt es zu jeder Klausel k_i, die y widerspricht, ein positives Beispiel x dem auch durch k_i widersprochen wird (sonst wäre diese Klausel nicht entfernt worden). Das bedeutet aber, daß es keine konsistente k-CNF_n Formel geben kann: WIDERSPRUCH!

Die Konsistenzprobleme für *Sym* und *Rechteck*$_d$ sind trivial. Den Beweis von c) geben wir Hier nicht an. □

Beachte, daß das Konsistenzproblem für die obigen Konzeptklassen in Zeit $poly\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n\right)$ gelöst wird: Eine Abhängigkeit von der minimalen Namenslänge ist nicht vorhanden. Eine solche Abhängigkeit tritt bei dem bekanntesten bisher ungelösten Problem des effizienten Lernens auf: Dem effizienten Lernen von DNF. Die polynomielle Lernbarkeit von DNF scheint unmöglich, da $VC(DNF_n) = 2^n$ ist, und somit $\Omega\left(\frac{1}{\varepsilon}(2^n + \ln\left(\frac{1}{\delta}\right)\right)$ Beispiele angefordert werden müssen! Diese Situation ändert sich aber für DNF-Formeln der Länge höchstens k: die VC-Dimension ist durch k beschränkt, da es nur 2^k verschiedene Konzepte gibt. Die VC-Dimension ist also kein Argument gegen die polynomielle Lernbarkeit.

3.1 Das Konsistenzproblem

Wie wichtig ist die Lösung des Konsistenzproblems? Ist es möglich, daß es effiziente PAC-Algorithmen gibt, die lernen ohne konsistent zu sein?

Sei L ein effizienter PAC-Algorithmus für eine parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$. Wir möchten, ausgehend von L, einen effizienten Algorithmus L^* konstruieren, der für jede Beispielsmenge S eine mit S konsistente Hypothese $c \in \mathcal{C}_n$ findet. Wie sieht L^* aus?
Es sei eine klassifizierte Beispielmenge S gegeben.

- L^* arbeitet probabilistisch.

- L^* liest die Beispielmenge S ein und generiert dann Beispiele gemäß der Verteilung

$$D(x) = \begin{cases} 0 & x \notin S \\ \frac{1}{|S|} & \text{sonst.} \end{cases}$$

(Wenn Beispiele gemäß D generiert werden, werden nur Beispiele aus S gewählt.)

- L^* ruft den deterministischen Lernalgorithmus L mit den Parametern $\varepsilon = \frac{1}{|S|+1}$ und $\delta = \frac{1}{2}$ und der **Folge** der gemäß D ausgewählten Beispiele auf.
- Der Lernalgorithmus L erreicht

$$prob[Fehler_D(c, h_c) \leq \frac{1}{|S|+1}] \geq \frac{1}{2};$$

  d.h. L liefert mit Wahrscheinlichkeit mindestens $1 - \delta = \frac{1}{2}$ eine Hypothese h_c mit

$$Fehler_D(c, h_c) = \sum_{\substack{x \in S \\ x \text{ falsch klassifiziert}}} D(x) = \sum_{\substack{x \in S \\ x \text{ falsch klassifiziert}}} \frac{1}{|S|} < \frac{1}{|S|+1}.$$

  Mit anderen Worten: L löst das Konsistenzproblem für S mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ effizient.
- L^* gibt h_c aus.

Wir erhalten also aus dem deterministischen Algorithmus L einen probabilistischen Algorithmus L^*, der das Konsistenzproblem für $\mathcal{C}$ mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ in Zeit

$$poly\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, n, \min\{|w| \mid w \in Name(c')\}\right) = poly\left(|S|, n, \min\{|w| \mid w \in Name(c')\}\right)$$

löst. (Hierbei ist c' ein beliebiges mit der Beispielsmenge konsistentes Konzept.)
Wir möchten Aussagen über die Komplexität des Konsistenzproblems machen und gehen deshalb vom Konsistenzproblem zu einer Sprachversion des Konsistenzproblems über:

3.1.1 Definition Sei $\mathcal{C}$ eine parametrisierte Konzeptklasse über Σ. Wir definieren die Sprache $Konsistenz_{\mathcal{C}}$ durch:

$$Konsistenz_{\mathcal{C}} = \{(n, x_1, b_1, \ldots, x_s, b_s) \mid \text{ Es gibt } \mathrm{c} \in \mathcal{C}_\mathrm{n} \text{ mit } (\mathrm{x_i} \in \mathrm{c} \Leftrightarrow \mathrm{b_i} = 1),\ 1 \leq \mathrm{i} \leq \mathrm{s}\},$$

wobei $n \in \mathbb{N}$, $x_i \in \Sigma^{\leq n}$ (für $1 \leq i \leq s$), $b_i \in \{0, 1\}$ (für $1 \leq i \leq s$) ist.

Die Laufzeit des oben konstruierten Algorithmus L^* ist polynomiell in der Größe der Beispielsmenge, der Eingabelänge und in der Namensfunktion. Wir werden als nächstes fordern, daß keine zu langen Namen vergeben werden.

3.1.2 Definition Sei $\mathcal{C}$ eine parametrisierte Konzeptklasse über Σ mit Namensfunktion $Name : \bigcup_{n=0}^{\infty} \mathcal{C}_n \rightarrow \mathcal{P}(\{0, 1\}^*)$. $Name$ heißt genau dann effizient, wenn es einen deterministischen, polynomiellen Algorithmus A gibt, der für alle $w \in \{0, 1\}^*$, für alle $n \in \mathbb{N}$ und für jede Eingabe $x \in \Sigma^*$ nachprüft, ob

- w der Name eines Konzeptes in $\mathcal{C}_n$ ist und,
- ob $x \in c$ für das Konzept $c \in \mathcal{C}_n$ mit $w \in Name(c)$ gilt.

Zusätzlich wird gefordert, daß es für jedes $n \in \mathbb{N}$ und für jedes Konzept $c \in \mathcal{C}_n$ ein $k \in \mathbb{N}$ gibt mit

$$\min\{|w| \mid w \in Name(c)\} \leq n^k.$$

Eine effiziente Namensfunktion vergibt also nicht zu lange Namen und erlaubt eine effiziente Lösung des Elementproblems $x \in c$. (Die obige Definition gilt für alle „vernünftigen“ Namensfunktionen; insbesondere sind alle bisher benutzten Namensfunktionen effizient.)

Beispiel Eine effiziente Namensfunktion für ein Konzept c aus der parametrisierten Konzeptklasse k-CNF, k konstant, ist:

$$Name(c) = \{ \text{ Binärkodierung einer } k\text{-}CNF \text{ Formel } \mid wahr(\alpha) = c\}.$$

(Ein Konzept besitzt somit viele Namen, nämlich die Binärkodierung aller äquivalenten k-CNF Formeln α mit $wahr(\alpha) = c$.) Warum ist diese Namensfunktion effizient?
Sei $w \in Name(c)$ mit $w \equiv$ Binärkodierung(α) gegeben. Da es höchstens $2n^k$ Klauseln der Länge $\leq k$ gibt, kann in polynomieller Zeit überprüft werden, ob w der Name eines Konzepts in $\mathcal{C}_n$ ist.
Für eine Eingabe $x \in \Sigma^{\leq n}$ kann effizient überprüft werden, ob $\alpha(x) = 1$ gilt: Es wird jede Klausel verifiziert.
Letztendlich ist $|w| = O((2n)^k \cdot k \cdot \underbrace{\log_2 n}_{\text{Name der Literale}}) = poly_k(n)$ und es werden keine zu langen Namen vergeben.
Die bisherigen Überlegungen führen zu folgendem Resultat:

3.1.3 Satz *Sei $\mathcal{C}$ eine parametrisierte Konzeptklasse mit effizienter Namensfunktion Name. Wenn es einen polynomiellen PAC-Algorithmus für $\mathcal{C}$ mit Name gibt, dann kann Konsistenz$_{\mathcal{C}}$ in polynomieller Zeit durch einen Monte-Carlo-Algorithmus berechnet werden.*

[Ein Monte-Carlos-Algorithmus für eine Sprache L ist ein probabilistischer Algorithmus, der sich bei positiver Antwort nie irrt und bei negativer Antwort eine Fehlerwahrscheinlichkeit von höchstens $\frac{1}{2}$ besitzt. Jeder Monte-Carlo-Algorithmus ist insbesondere ein nichtdeterministischer Algorithmus.]

Beweis Sei L ein polynomieller PAC-Algorithmus für $\mathcal{C}$. Entwurf eines Monte-Carlo-Algorithmus L^{**}:

- L^{**} erhält eine Eingabe $(n, x_1, b_1, \ldots, x_s, b_s)$.
- L^{**} konstruiert daraus die Beispielmenge $S = \{(x_i, b_i) \mid 1 \leq i \leq s\}$.

- L^{**} ruft den (auf Seite 43 beschriebenen) probabilistischen Algorithmus L^* mit der Eingabe $(n, x_1, b_1, \ldots, x_s, b_s)$ auf.
- L^* liefert eine Hypothese h.
- L^{**} überprüft, ob h mit der Beispielsmenge S konsistent ist. Falls h mit S konsistent ist, akzeptiert L^{**}; sonst verwirft er.

Analyse von L^{**} **Fall 1** $(n, x_1, b_1, \ldots, x_s, b_s) \in Konsistenz_{\mathcal{C}}$, d.h. gibt eine mit S konsistente Hypothese. Dann liefert L^* mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ eine mit S konsistente Hypothese. Somit akzeptiert L^{**} mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ und verwirft mit Wahrscheinlichkeit höchstens $\frac{1}{2}$.
Fall 2 $(n, x_1, b_1, \ldots, x_s, b_s) \notin Konsistenz_{\mathcal{C}}$, d.h. es gibt **keine** mit S konsistente Hypothese. Dann liefert L^* stets eine mit S **nicht** konsistente Hypothese ab. Somit verwirft L^{**} stets.
Konsequenz der beiden Fälle: L^{**} ist ein Monte-Carlo-Algorithmus, da er bei positiver Antwort nie irrt und bei negativer Antwort mit Wahrscheinlichkeit höchstens $\frac{1}{2}$ irrt.
Laufzeit von L^{**} L^{**} läuft (unter der Voraussetzung, daß $c \in \mathcal{C}_n$ ist) in Zeit

$$poly(n, |S|, \min\{|w| \mid w \in Name(c)\}) = poly(n, |S|),$$

da $Name$ effizient ist. (L^{**} dekodiert zunächst die Eingabe. Dann wird die wesentliche Arbeit von L^* geleistet, der Zeit in $poly(n, |S|)$ arbeitet. Als letztes überprüft L^{**}, ob die von L^* gelieferte Hypothese konsistent mit S ist; dies ist in Zeit $poly(n, |S|)$ möglich, da $|S|$ Beispiele der Eingabelänge n mit h verglichen werden.)

□

Angenommen $Konsistenz_{\mathcal{C}}$ ist NP-vollständig. Dann ist es nicht zu erwarten, daß ein polynomieller PAC-Algorithmus für $\mathcal{C}$ existiert. Andererseits schließt die Annahme $P \neq NP$ einen polynomiellen PAC-Algorithmus nicht aus, denn wir betrachten nicht **deterministische** Algorithmen sondern die mächtigeren **Monte-Carlo** Algorithmen. Wir benötigen somit eine stärkere Annahme als $P \neq NP$.

3.1.4 Definition R ist die Klasse aller Sprachen, die durch einen polynomiellen Monte-Carlo-Algorithmus berechnet werden können.

Es ist $P \subseteq R \subseteq NP$. Die genauen Beziehungen sind ein offenes Problem. Es gilt weiterhin, daß $R = NP$, falls es ein NP-vollständiges Problem L mit $L \in R$ gibt.

3.1.5 Satz *Angenommen $R \neq NP$. Sei $\mathcal{C}$ eine parametrisierte Konzeptklasse mit effizienter Namensfunktion $Name$. Wenn $Konsistenz_{\mathcal{C}}$ NP-vollständig ist, dann gibt es* **keinen** *polynomiellen PAC-Algorithmus für $\mathcal{C}$ (und Namensfunktion $Name$), der $\mathcal{C}$ auch als Hypothesenklasse benutzt.*

(Mit anderen Worten: Falls $R \neq NP$ gilt, dann ist die Komplexität des Konsistenzproblems dafür verantwortlich, daß es keinen polynomiellen PAC-Algorithmus gibt.)

Beweis Angenommen L ist ein polynomieller PAC-Algorithmus für $\mathcal{C}$ mit effizienter Namensfunktion $Name$. Mit Satz 3.1.3 wissen wir, daß es dann einen polynomiellen Monte-Carlo-Algorithmus für $Konsistenz_{\mathcal{C}}$ gibt. Somit gilt $Konsistenz_{\mathcal{C}} \in R$. Wenn $Konsistenz_{\mathcal{C}}$ NP-vollständig ist, folgt $R = NP$ im WIDERSPRUCH zu der Annahme $R \neq NP$. □

Mit Satz 3.1.5 können wir negative Aussagen für repräsentations-abhängiges Lernen (also Lernen, bei dem die Konzeptklasse der Hypothesenklasse entspricht) machen. Ein erstes Beispiel ist k-Klausel CNF (und damit auch k-Term DNF).

3.1.6 Satz *Angenommen es gilt $R \neq NP$. Dann gibt es* **keinen** *polynomiellen PAC-Algorithmus, der die Konzeptklasse k-Klausel CNF (mit der Binärkodierung der k-Klausel CNF Formel als Namensfunktion) durch die Hypothesenklasse k-Klausel CNF erlernt.*

Beweis Wir müssen zeigen, daß $Konsistenz_{k\text{-Klausel}\,CNF}$ NP-vollständig ist. Dazu reduzieren wir das NP-vollständige Problem 3-$FÄRBBARKEIT$ auf das Sprachproblem $Konsistenz_{3\text{-Klausel}\,CNF}$.
(3-$FÄRBBARKEIT$: Gegeben sei ein ungerichteter Graph G mit Knotenmenge V und Kantenmenge E. G ist genau dann 3-färbbar, wenn V so mit drei Farben gefärbt werden kann, daß die Endknoten jeder Kante aus E verschiedene Farben haben.)

Behauptung 3-$FÄRBBARKEIT \leq_p Konsistenz_{3\text{-Klausel}\,CNF}$,
wobei $\leq_p$ die Notation für „polynomiell reduzierbar auf" ist.

Beweis Zuerst eine Beschreibung der Reduktion: Sei der ungerichtete Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$ die Eingabe für 3-$FÄRBBARKEIT$. Wir konstruieren als Eingabe von $Konsistenz_{3\text{-Klausel}\,CNF}$ die Beispielsmenge

$$S = \{(e_v, 0) \mid 1 \leq v \leq n\} \cup \{(e_v + e_w, 1) \mid \{v, w\} \in E\}$$

(wobei $e_v = (0 \ldots 0 \underbrace{1}_{\text{Pos. } v} 0 \ldots 0)$ mit $|e_v| = n$ ist).

Wir müssen jetzt noch nachweisen, daß G genau dann 3-färbbar ist, wenn es eine mit S konsistente 3-Klausel CNF gibt:

Angenommen der Graph G ist 3-färbbar mit der Färbung $Farbe : V \to \{1, 2, 3\}$.
Wir konstruieren die Formel $c = k_1 \wedge k_2 \wedge k_3$, mit

$$k_i \equiv \bigvee_{\substack{v \in V \\ Farbe(v) \neq i}} x_v$$

(also besteht z. B. k_1 aus all den positiven Literalen x_v $(1 \leq v \leq n)$, für die Knoten v **nicht** mit 1 gefärbt ist).

Diese 3-Klausel CNF_n Formel c ist mit S konsistent:
c verwirft jede Eingabe e_v (mit der Klausel, die der Farbe von v entspricht): Es ist $c(e_v) = k_1(e_v) \wedge k_2(e_v) \wedge k_3(e_v)$. Sei o.B.d.A. $Farbe(v) = 1$, dann ist $k_1(e_v) = 0$ und somit $c(e_v) = 0$.
c akzeptiert jede Eingabe $e_v + e_w$, mit $(v, w) \in E$: Es ist $c(e_v + e_w) = k_1(e_v + e_w) \wedge k_2(e_v + e_w) \wedge k_3(e_v + e_w)$. Da $\{v, w\} \in E$ und G 3-färbbar ist, besitzen v und w verschiedene Farben. Somit enthält **jede** Klausel entweder x_v oder x_w, und es gilt $k_1(e_v + e_w) = k_2(e_v + e_w) = k_3(e_v + e_w) = 1$.

Angenommen die 3-Klausel CNF_n Formel $c \equiv k_1 \wedge k_2 \wedge k_3$ ist konsistent mit der Beispielsmenge S. Dann gibt es für jedes e_v, $1 \leq v \leq n$, mindestens eine Klausel $k_{f(v)}$ mit $k_{f(v)}(e_v) = 0$ (d.h. das Literal x_v kommt in $k_{f(v)}$ nicht vor).
Wir wählen $f(v) \in \{1, 2, 3\}$ als Farbe für Knoten v. Sei $\{v, w\} \in E$ eine Kante. Dann gilt $f(w) \neq f(v)$: Sonst enthält die Klausel $k_{f(v)}$ weder das Literal x_v noch das Literal x_w, und $e_v + e_w$ ist ein negatives Beispiel (denn aus $k_{f(v)}(e_v + e_w) = 0$ folgt $c(e_v + e_w) = 0$). Also ist f ein 3-Färbung von G ■

Wir wissen bereits aus Satz 3.3, daß die Konzeptklasse k-Term DNF mit k-CNF als Hypothesenklasse polynomiell lernbar ist. Satz 3.1.6 drückt somit aus, daß die Wahl der Hypothesenklasse für die effiziente Lösung des Konsistenzproblems von großer Bedeutung ist.

3.2 Occam-Algorithmen

Der Philosoph William of Ockham (1295 - 1349) hat das folgende Prinzip aufgestellt (auch Occam's Razor genannt): *„Non sunt multiplicanda entia praeter necessitatem“*
(Objekte sind nicht unnötigerweise zu multiplizieren. Oder, in freier Übersetzung: Wähle stets die einfachste Erklärung.) Auf Lernalgorithmen angewendet, wählen wir die folgende Version: *„Wähle stets eine Hypothese mit* **kurzer** *Beschreibungslänge“.*
Die Beschreibungslänge einer Hypothese ist dabei die Länge des Names der Hypothese. Die Forderung nach einer Hypothese mit **kürzester** Beschreibungslänge erscheint sinnvoller, doch lassen sich kürzeste Hypothesen oft nicht effizient konstruieren.

3.2.1 Definition Ein Lernalgorithmus L für eine Hypothesenklasse $\mathcal{H}$ mit Namensfunktion $Name : \mathcal{H} \rightarrow \mathcal{P}(\{0,1\}^*)$ heißt genau dann Occam-Algorithmus für α $(0 \leq \alpha < 1)$ und eine Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$, wenn L die folgende Eigenschaft besitzt:
Für jede klassifizierte Beispielsmenge S (mit $|S| = s$), für jede mit S konsistente Hypothese $h \in \mathcal{H}$ und für jedes t, mit $|Name(h)| < t$, produziert L eine mit S **konsistente** Hypothese $h' \in \mathcal{H}$ mit

$$|Name(h')| \leq f(t) \cdot s^{\alpha}.$$

3.2.2 Satz *Sei L ein Occam-Algorithmus für α $(0 \leq \alpha < 1)$ und f. Dann ist L ein PAC-Algorithmus, der mit*

$$s \geq \left(\frac{2 \cdot f(t) \cdot \ln(2)}{\varepsilon}\right)^{\frac{1}{1-\alpha}} + \frac{2}{\varepsilon} \cdot \ln\left(\frac{1}{\delta}\right)$$

Beispielen auskommt (für zu lernende Konzepte der Beschreibungslänge t).

Beweis Wir setzen $\mathcal{H}_r = \{h \in \mathcal{H} \mid |Name(h)| < r\}$. Dann gilt offensichtlich $|\mathcal{H}_r| \leq 2^r - 1$. Sei $p_{s,r}$ die Wahrscheinlichkeit, daß es eine Hypothese $h \in \mathcal{H}_r$ gibt, die mit s Beispielen (generiert gemäß der unbekannten Verteilung D) konsistent ist und einen Fehler größer als ε besitzt. Wir müssen nachweisen, daß $p_{s,r} \leq \delta$ für $r = f(t) \cdot s^\alpha$ gilt. Es ist, wie wir schon im Beweis von Satz 2.2.1 gesehen haben,

$$\begin{aligned} p_{s,r} &\leq (1-\varepsilon)^s \cdot |\mathcal{H}_r| \\ &\leq (1-\varepsilon)^s \cdot 2^r. \end{aligned}$$

Wir müssen s so groß wählen, daß $p_{s,r} \leq \delta$ gilt. Dies führt nach Logarrithmierung zu:

$$s \cdot \ln(1-\varepsilon) + r \cdot \ln(2) \leq \ln(\delta)$$

beziehungsweise, unter Ausnutzung von $\ln(1-\varepsilon) \leq -\varepsilon$, zu:

$$\begin{aligned} & -s \cdot \varepsilon + r \cdot \ln(2) &\leq\ & \ln(\delta) \\ \Leftrightarrow\quad & s &\geq\ & \tfrac{1}{\varepsilon} \cdot r \cdot \ln(2) + \tfrac{1}{\varepsilon} \cdot \ln\left(\tfrac{1}{\delta}\right). \qquad (*) \end{aligned}$$

Wir setzen nun $r = f(t) \cdot s^\alpha$ ein (gemäß Definition des Occam-Algorithmus wird L eine Hypothese in $\mathcal{H}_r$ finden).

(1) Wegen der Voraussetzungen des Satzes können wir für s fordern:

$$\begin{aligned} & s &\geq\ & \left(\tfrac{2 \cdot f(t) \cdot \ln(2)}{\varepsilon}\right)^{\frac{1}{1-\alpha}} \\ \Leftrightarrow\quad & \tfrac{s}{s^\alpha} &\geq\ & \tfrac{2 \cdot f(t) \cdot \ln(2)}{\varepsilon} \\ \Leftrightarrow\quad & \tfrac{s}{2} &\geq\ & s^\alpha \cdot \tfrac{f(t) \cdot \ln(2)}{\varepsilon} = \tfrac{1}{\varepsilon} \cdot r \cdot \ln(2). \end{aligned}$$

(2) Außerdem gilt aufgrund der Voraussetzungen des Satzes

$$\begin{aligned} & s &\geq\ & \tfrac{2}{\varepsilon} \cdot \ln\left(\tfrac{1}{\delta}\right) \\ \Leftrightarrow\quad & \tfrac{s}{2} &\geq\ & \tfrac{1}{\varepsilon} \cdot \ln\left(\tfrac{1}{\delta}\right). \end{aligned}$$

Aus (1) und (2) folgt die geforderte Ungleichung $(*)$. □

Da $s \geq \left(\frac{2 \cdot f(t) \cdot \ln(2)}{\varepsilon}\right)^{\frac{1}{1-\alpha}} \geq f(t)^{\frac{1}{1-\alpha}}$ ist, gilt $f(t) < s^{1-\alpha}$ beziehungsweise

$$\underbrace{f(t) \cdot s^{\alpha}}_{\text{Länge der konsistenten Hypothese}} < \underbrace{s}_{\text{Anzahl Beispiele}}.$$

Mit anderen Worten: Occam-Algorithmen führen eine Art von Datenkomprimierung durch: s Beispiele werden durch eine Hypothese der Länge $f(t) \cdot s^{\alpha} < s$ repräsentiert.

Wir kommen jetzt zu einer ersten Anwendung von Occam-Algorithmen.

Beispiel Wir entwerfen einen Occam-Algorithmus für die Konzeptklasse $Monom_n$. Wir nehmen an, daß der Name des Monoms

$$m = l_1 \wedge \ldots \wedge l_k$$

durch die Binärdarstellung

$$b_1\ bin(l_1)\ b_2\ bin(l_2) \ldots b_k\ bin(l_k)$$

gegeben ist. Hierbei gibt $b_i \in \{0,1\}$ $(1 \leq i \leq k)$ an, ob das Literal l_i positiv ($b_i = 0$) oder negativ ($b_i = 1$) vorkommt. $bin(l_i)$ $(1 \leq i \leq k)$ ist die Binärdarstellung der Variablennummer-1 des Literals l_i. Die Binärdarstellung ist dabei mit führenden Nullen auf Länge $\lceil \log_2 n \rceil$ aufgefüllt worden. (Z.B.: Sei $n = 8$ und $l_1 = \neg x_4$, dann ist $bin(l_1) = 011$.)

Hypothesen mit kurzem Namen entsprechen Monomen mit wenigen Literalen. Der zu entwerfende Occam Algorithmus muß also ein konsistentes Monom mit möglichst wenigen Literalen finden.

Sei $S \subseteq \{0,1\}^n$ eine klassifizierte Beispielsmenge mit $S = S_- \cup S_+$. S_- (S_+) ist die Menge der negativen (positiven) Beispiele. Wir betrachten die Menge L aller Literale, die für **alle** positiven Beispiele den Wert 1 ergeben. Also ist

$$L = \{l \mid l \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} \text{ und } l(y) = 1 \text{ für alle y} \in \text{S}_+\}.$$

(Dann ist $m = \bigwedge_{l \in L} l$ ein (bzgl. obiger Namensfunktion) längstes mit S konsistentes Monom.) Um ein kurzes Monom zu erhalten, das mit S konsistent ist, versuchen wir alle negativen Beispiele mit möglichst wenigen Literalen aus L auszuschalten. (Diese möglichst wenigen Literale bilden dann unser Monom.)

Für jedes $l \in L$ sei

$$negativ(l) = \{y \in S_- \mid l(y) = 0\}.$$

Dann ist

$$S_- = \bigcup_{l \in L} negativ(l),$$

und wir suchen eine möglichst kleine Menge $L' \subseteq L$ mit

$$S_- = \bigcup_{l \in L'} negativ(l).$$

Wir betrachten das Überdeckungsproblem, eine Abstraktion unseres Problems. Im Überdeckungsproblem sind ein Universum U und Mengen $A_1, \ldots, A_n \subseteq U$ gegeben, wobei $A_1 \cup \ldots \cup A_n = U$ ist. Eine Menge $I \subseteq \{1, \ldots, n\}$ kleinster Größe ist gesucht, so daß

$$\bigcup_{i \in I} A_i = U$$

gilt.

Das Überdeckungsproblem ist NP-vollständig, aber es existieren effiziente Approximationsalgorithmen. Wir beschreiben den gierigen Überdeckungsalgorithmus, der eine Überdeckung mit höchstens

$$opt \cdot (\ln(|U|) + 1)$$

Mengen findet, wobei *opt* die minimale Anzahl an überdeckenden Mengen ist (siehe D.S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences, 9, pp. 256-278, 1974):
Anfangs sei $I := \emptyset$. Zuerst wird eine größte Menge A_i zur Überdeckung gewählt (der gierige Schritt) und $I = I \cup \{i\}$ gesetzt. Danach werden alle Elemente von A_i aus den Mengen $A_1, \ldots, A_n$ entfernt (d.h. A_j wird durch $A_j \setminus A_i$ ersetzt).
Diese beiden Schritte (Bestimmung der größten Menge und ihre Entfernung) werden wiederholt, bis alle Elemente aus U entfernt sind (und damit $\bigcup_{i \in I} A_i = U$ ist).

Wir wenden den gierigen Überdeckungsalgorithmus auf das Universum S_- und die Mengen $negativ(l)$, für $l \in L$, an und erhalten damit ein mit S konsistentes Monom mit höchstens

$$opt \cdot (\ln(|S_-|) + 1)$$

Literalen (wobei *opt* die minimale Literalanzahl für ein mit S konsistentes Monom ist). Wir finden also ein mit S konsistentes Monom mit Namenslänge

$$opt \cdot (\lceil \log_2 n \rceil + 1) \cdot (\ln(|S_-|) + 1).$$

Wir beschränken uns auf $\varepsilon \geq \frac{1}{n}$ und $\delta \geq 2^{-n}$. Dann wird die von uns angeforderte Beispielsanzahl n^2 nicht überschreiten und es gilt $\ln(|S_-|) \leq 2\ln(n)$. Wir haben also einen Occam-Algorithmus für

$$\alpha = 0 \quad \text{und} \quad \mathrm{f(t)} = \lceil \mathrm{t} \cdot (2 \cdot \ln(\mathrm{n}) + 1) \rceil$$

erhalten. Nach Satz 3.2.2 genügt es

$$O\left(\frac{1}{\varepsilon}\left(opt \cdot (\ln(n) + 1) + \ln\left(\frac{1}{\delta}\right)\right)\right)$$

Beispiele anzufordern. (Beachte, daß die Beispielzahl durch n^2 beschränkt ist).

Unser Occam-Algorithmus läuft in polynomieller Zeit. Seine **wesentliche** Eigenschaft ist, daß sich der Algorithmus der Komplexität des Zielkonzeptes anpaßt: Wenige Beispiele genügen, um Monome mit wenigen Literalen zu lernen.

Vorteil der Occam-Algorithmen ist also, daß einfache Konzepte mit entsprechend wenigen Beispielen gelernt werden.

Wir kommen jetzt zu einer zweiten Anwendung von Occam-Algorithmen.

3.2.3 Definition Sei $\mathcal{C}$ eine Konzeptklasse.

a) Wir definieren die t-fache Vereinigung von $\mathcal{C}$ durch $V_t(\mathcal{C}) = \{c_1 \cup \ldots \cup c_t \mid c_1, \ldots, c_t \in \mathcal{C}\}$ und die beliebige Vereinigung von $\mathcal{C}$ durch $V(\mathcal{C}) = \bigcup_{t=1}^{\infty} V_t(\mathcal{C})$.

b) Analog definieren wir den t-fachen Durchschnitt von $\mathcal{C}$ durch $D_t(\mathcal{C}) = \{c_1 \cap \ldots \cap c_t \mid c_1, \ldots, c_t \in \mathcal{C}\}$ und den beliebigen Durchschnitt von $\mathcal{C}$ durch $D(\mathcal{C}) = \bigcap_{t=1}^{\infty} D_t(\mathcal{C})$.

Beispiele $V(Rechteck_2)$ ist die Konzeptklasse aller Vereinigungen von Rechtecken in $\mathbb{R}^2$. $D(Threshold_2)$ ist die Konzeptklasse aller konvexen Polygone in $\mathbb{R}^2$.

Sei $\mathcal{C}^*$ eine Konzeptklasse mit endlicher VC-Dimension. Unser Ziel ist die Konstruktion von polynomiellen PAC-Algorithmen sowohl für $V(\mathcal{C}^*)$ als auch für $D(\mathcal{C}^*)$. Dafür bieten sich Occam-Algorithmen an: Für ein unbekanntes Konzept $c_1 \cup \ldots \cup c_t \in V(\mathcal{C}^*)$ ist eine Hypothese $h \in V(\mathcal{C}^*)$ zu finden, so daß h eine Vereinigung von **möglichst wenigen** Konzepten aus $\mathcal{C}^*$ ist. h läßt sich somit als eine kurze Erklärung verstehen.
Die allgemeine Theorie wird uns helfen, PAC-Algorithmen für Probleme wie „die Vereingung von Rechtecken in $\mathbb{R}^2$“ und „konvexe Polygone in $\mathbb{R}^2$“ zu entwerfen.

Zuerst formulieren wir den Begriff eines Occam-Algorithmus neu:

3.2.4 Definition Sei $\mathcal{C} = (\mathcal{C}_t \mid t \in \mathbb{N})$ eine parametrisierte [1] Konzeptklasse und $\mathcal{H}$ eine Hypothesenklasse. L sei ein **konsistenter** Lernalgorithmus.

a) Der Hypothesenraum $\mathcal{D}_{t,s}$ (für s Beispiele und ein zu lernendes Konzept aus $\mathcal{C}_t$) besteht aus allen möglichen Hypothesen aus $\mathcal{C}$, die von L aufgestellt werden, wenn L eine mit einem Konzept aus $\mathcal{C}_t$ konsistente Menge von s Beispielen erhält:

$$\mathcal{D}_{t,s} = \{h \mid h \in \mathcal{H},\ \ L \text{ liefert } h, \text{ falls } L \text{ eine mit einem } c \in \mathcal{C}_t \text{ konsistente Beispielmenge der Größe } s \text{ erhält}\}.$$

b) L heißt Occam-Algorithmus für α $(0 \leq \alpha < 1)$ und die Funktion f, falls $VC(\mathcal{D}_{t,s}) < f(t) \cdot s^{\alpha}$ gilt.

Wir gehen jetzt also (an Stelle der Namensgebung) von einem gegebenem Komplexitätmaß aus, das jedem Konzept in $c \in \mathcal{C}$ das kleinste t mit $c \in \mathcal{C}_t$ zuweist. Dieses Vorgehen ist zum Beispiel für Konzeptklassen kleiner VC-Dimension angebracht, wo unnötig lange Namen vergeben werden. Ein Beispiel ist die Klasse der konvexen Polygone im $\mathbb{R}^2$: Ein Name eines konvexen Polygons wird im allgemeinen sehr viel länger sein als die Anzahl der

[1] Die Parametrisierung erfolgt nicht notwendigerweise nach der Eingabelänge: Es wird nicht verlangt, daß $c \in \mathcal{C}_t \Rightarrow c \in \Sigma^{\leq t}$ gilt.

Eckpunkte (als neues Komplexitätsmaß). Wir werden im folgenden eine Parametrisierung durch die Anzahl von Vereinigungen betrachten, also $\mathcal{C} = (V_t(\mathcal{C}^*) \mid t \in \mathbb{N})$.

Scheinbar haben wir einen neuen Typ von Occam-Algorithmus definiert. Dem ist nicht ganz so:

Bemerkung Sei L ein „alter" Occam-Algorithmus für die Konzeptklasse $\mathcal{C}$. Wir setzen $\mathcal{C}_t = \{c \in \mathcal{C} \mid \textit{länge}(c) < t\}$. Dann liefert L, für ein Zielkonzept $c \in \mathcal{C}_t$, eine Hypohese h der Länge höchstens $f(t) \cdot s^\alpha$ für s klassifizierte Beispiele. Somit ist $\mathcal{D}_{t,s} \subseteq \mathcal{C}_{f(t)\cdot s^\alpha}$.

Für die VC-Dimension erhalten wir dann $VC(\mathcal{D}_{t,s}) \leq VC(\mathcal{C}_{f(t)\cdot s^\alpha}) \leq \log_2 |\mathcal{C}_{f(t)\cdot s^\alpha}| \leq f(s) \cdot m^\alpha$, denn $|\mathcal{C}_i| \leq 2^0 + 2^1 + \ldots + 2^{i-1} = 2^i - 1$. Also ist ein alter Occam-Algorithmus für $\mathcal{C}$ ein neuer Occam-Algorithmus für $(\mathcal{C}_t \mid t \in \mathbb{N})$.

Wir zeigen zuerst, daß ein „neuer" Occam-Algorithmus stets ein PAC-Algorithmus ist.

3.2.5 Satz *Sei $\mathcal{C} = (\mathcal{C}_t \mid t \in \mathbb{N})$ eine Konzeptklasse. L sei ein Occam-Algorithmus mit $VC(\mathcal{D}_{t,s}) \leq f(t) \cdot s^\alpha$. Dann ist L ein PAC-Algorithmus, falls mindestens*

$$s \geq \max\left\{\left(\frac{128}{\varepsilon} \cdot f(t) \cdot \ln\left(\frac{6}{\varepsilon}\right)\right)^{\frac{1}{1-\alpha}}, \frac{32}{\varepsilon} \cdot \ln\left(\frac{2}{\delta}\right)\right\}$$

Beispiele angefordert werden.

Beweis Sei $p_{t,s}$ die Wahrscheinlichkeit, daß es eine Hypothese $h \in \mathcal{D}_{t,s}$ gibt, die mit s (gemäß der unbekannten Verteilung D generierten) Beispielen konsistent ist und einen Fehler größer als ε besitzt. Eine Beispielsanzahl s ist zu bestimmen, so daß $p_{t,s} \leq \delta$ gilt.

Im Beweis von Satz 2.4.2 haben wir gesehen, daß

$$p_{t,s} \leq 2\left(\frac{e \cdot 2s}{d}\right)^d \cdot 2^{-\frac{\varepsilon s}{2}}$$

gilt, wobei

$$d := VC(\mathcal{D}_{t,s}) < f(t) \cdot s^\alpha$$

ist. Die Ungleichung $p_{s,m} \leq \delta$ wird im Beweis von Satz 2.4.2 durch die Wahl von $s \geq \max\left\{\frac{128}{\varepsilon} \cdot d \cdot \ln\left(\frac{6}{\varepsilon}\right), \frac{32}{\varepsilon} \cdot \ln\left(\frac{2}{\delta}\right)\right\}$ erfüllt. [2]. Wenn wir d durch die obere Schranke $f(t) \cdot s^\alpha$ ersetzen, folgt

$$s \geq \max\left\{\frac{128}{\varepsilon} \cdot f(t) \cdot s^\alpha \cdot \ln\left(\frac{6}{\varepsilon}\right), \frac{32}{\varepsilon} \cdot \ln\left(\frac{2}{\delta}\right)\right\}.$$

[2] Tatsächlich haben wir $s \geq \frac{128}{\varepsilon} \cdot d \cdot \ln\left(\frac{6}{\varepsilon}\right) + \frac{32}{\varepsilon} \cdot \ln\left(\frac{2}{\delta}\right)$ gezeigt. Die gleichen Argumente beweisen aber auch $s \geq \max\left\{\frac{128}{\varepsilon} \cdot d \cdot \ln\left(\frac{6}{\varepsilon}\right), \frac{32}{\varepsilon} \cdot \ln\left(\frac{2}{\delta}\right)\right\}$.

Es ist

$$\begin{aligned} s &\geq \tfrac{128}{\varepsilon} \cdot f(t) \cdot s^{\alpha} \cdot \ln\left(\tfrac{6}{\varepsilon}\right) \\ \Leftrightarrow \quad s^{1-\alpha} &\geq \tfrac{128}{\varepsilon} \cdot f(t) \cdot \ln\left(\tfrac{6}{\varepsilon}\right) \\ \Leftrightarrow \quad s &\geq \left(\tfrac{128}{\varepsilon} \cdot f(t) \cdot \ln\left(\tfrac{6}{\varepsilon}\right)\right)^{\frac{1}{1-\alpha}}. \end{aligned}$$

□

Wir können jetzt polynomielle PAC-Algorithmen für $V(\mathcal{C})$ (und $D(\mathcal{C})$) entwerfen, falls $\mathcal{C}$ bestimmte Eigenschaften hat:

3.2.6 Satz *Sei die Konzeptklasse $\mathcal{C}$, mit $VC(\mathcal{C}) < \infty$ gegeben. Wenn es einen polynomiellen Algorithmus gibt, der für jede Beispielsmenge S eine mit S konsistente Hypothese aus $\mathcal{C}$ in Zeit poly($|S|$) findet, dann sind die Konzeptklassen $V(\mathcal{C})$ und $D(\mathcal{C})$ in* **polynomieller Zeit** *durch einen PAC-Algorithmus lernbar.*

Polynomielle Zeit bedeutet, daß der Lernalgorithmus in poly$(n, t, \frac{1}{\varepsilon}, \frac{1}{\delta})$ Schritten läuft, wobei n eine obere Schranke für die Eingabelänge sei.

Beweis Wir beschränken uns auf das Lernen der Konzeptklasse $V(\mathcal{C})$. (Der Beweis für $D(\mathcal{C})$ erfolgt analog.) Der Lernalgorithmus für $V(\mathcal{C})$ unter der Annahme, daß das Zielkonzept c zu $V_t(\mathcal{C})$ gehört, d.h. $c = c_1 \cup \ldots \cup c_t$, arbeitet wie folgt:

- Fordere eine Menge $S = \{(x_1, b_1), \ldots, (x_s, b_s)\}$ von s klassifizierten Beispielen an (wobei $s = poly(\frac{1}{\varepsilon}, \frac{1}{\delta}, t)$ ist).
- Berechne $F_{\{x_1,\ldots,x_s\}} = \{c \cap \{x_1, \ldots, x_s\} \mid c \in \mathcal{C}\}$.
- Finde (mit dem gierigen Überdeckungsalgorithmus) eine Überdeckung aller positiven Beispiele $S_+ = \{(x_i, b_i) \mid (x_i, b_i) \in S,\ b_i = 1\}$ mit möglichst wenigen Teilmengen aus $F_{\{x_1,\ldots,x_s\}}$. Eine jede solche Teilmenge darf nur positiv klassifizierte Beispiele enthalten.
- Gib die gefundene Überdeckung (also die Vereinigung der Konzepte) als Hypothese aus.

Analyse der Occam-Eigenschaft: Im dritten Schritt garantieren wir, daß der Lernalgorithmus konsistent ist. Wie „kurz" sind unsere Hypothesen?

Wenn das zu lernende Konzept c zur Klasse $V_t(\mathcal{C})$ gehört (also $c = c_1 \cup \ldots \cup c_t$), findet unser Lernalgorithmus eine Hypothese $h \in V_{t\cdot(\ln(s)+1)}(\mathcal{C})$ (denn unser Universum besteht aus maximal s Beispielen). Also ist $\mathcal{D}_{t,s} \subseteq V_{t\cdot(\ln(s)+1)}(\mathcal{C})$ und somit $VC(\mathcal{D}_{t,s}) \leq VC(V_{t\cdot(\ln(s)+1)}(\mathcal{C}))$.

Wir müssen also noch die VC-Dimension von $V_{t\cdot(\ln(s)+1)}(\mathcal{C})$ bestimmen:

3.2.7 Lemma Es gilt

$$VC(V_t(\mathcal{C})) \leq 2 \cdot t \cdot VC(\mathcal{C}) \cdot \log_2(4t).$$

Beweis Das Lemma wird in Aufgabe 4.3 bewiesen. □

Damit erhalten wir, für $s \geq e$ und $t \geq 1$,

$$\begin{aligned} VC(\mathcal{D}_{t,s}) &\leq 2 \cdot t \cdot (\ln(s)+1) \cdot VC(\mathcal{C}) \cdot \log_2(4t(\ln(s)+1)) \\ &\leq 2 \cdot t \cdot (\ln(s)+1) \cdot VC(\mathcal{C}) \cdot \log_2(4t) \cdot \log_2(\ln(s)+1), \\ &\leq 2 \cdot t \cdot \log_2(4t) \cdot VC(\mathcal{C}) \cdot (\ln(s)+1) \cdot (\ln(s)+1) \\ &\leq 2 \cdot t \cdot \log_2(4t) \cdot VC(\mathcal{C}) \cdot (2 \cdot \ln(s)) \cdot (2 \cdot \ln(s)) \\ &= \underbrace{8t \cdot \log_2(4t) \cdot VC(\mathcal{C})}_{=:f(t)} \cdot \underbrace{\ln^2(s)}_{=:s^\alpha}, \end{aligned}$$

mit $\alpha = \frac{2 \cdot \ln\ln(s)}{\ln(s)} < 1$. Wir können somit Satz 3.2.5 anwenden und erhalten, daß eine Beispielsanzahl von

$$s = \max\left\{ \left(\frac{128}{\varepsilon} \cdot f(t) \cdot \ln\left(\frac{6}{\varepsilon}\right)\right)^{\frac{1}{1-\alpha}}, \; \frac{32}{\varepsilon} \cdot \ln\left(\frac{2}{\delta}\right) \right\}$$

ausreicht. Wähle s so groß, daß $\alpha \leq \frac{1}{2}$ gilt. Dann genügen

$$poly(\frac{1}{\varepsilon}, t, \frac{1}{\delta})$$

viele Beispiele. Damit ist unser Algorithmus ein PAC-Algorithmus.

Zeitanalyse: Es genügt zu zeigen, daß die Menge $F_{\{x_1,\ldots,x_s\}}$ in Zeit $poly(s)$ bestimmt werden kann, und, daß der gierige Überdeckungsalgorithmus in Zeit $poly(s)$ läuft.

3.2.8 Lemma Es gilt:
a) $|F_{\{x_1,\ldots,x_s\}}| \leq \left(\frac{e \cdot s}{d}\right)^d = poly_1(s)$, wobei $d = VC(\mathcal{C})$ ist.
b) Es gibt einen Algorithmus, der alle Mengen aus $F_{\{x_1,\ldots,x_s\}}$ in Zeit $poly_2(s)$ aufzählt.

Beweis a) Siehe Aufgabe 4.2 b).
b) Wir konstruieren die Menge $F_{\{x_1,\ldots,x_s\}}$ rekursiv. Offensichtlich ist $F_{\{\emptyset\}} = \{\emptyset\}$. Angenommen $F_{\{x_1,\ldots,x_i\}}$ ist berechnet. Wir konstruieren daraus $F_{\{x_1,\ldots,x_i,x_{i+1}\}}$.
Sei $T \in F_{\{x_1,\ldots,x_i\}}$. Überprüfe in Zeit $poly(s)$, ob es ein Konzept $c \in \mathcal{C}$ gibt, das mit der positiven Beispielsmenge T und der negativen Beispielsmenge $\{x_1,\ldots,x_{i+1}\} \setminus T$ konsistent ist. (Ein solcher Algorithmus existiert nach Voraussetzung.) Wenn ja, füge die Menge T zu $F_{\{x_1,\ldots,x_i,x_{i+1}\}}$ hinzu. Wenn nein, ist $T \notin F_{\{x_1,\ldots,x_i,x_{i+1}\}}$.
Überprüfe in Zeit $poly(s)$, ob es ein Konzept $c \in \mathcal{C}$ gibt, das mit der positiven Beispielsmenge $T \cup \{x_{i+1}\}$ und der negativen Beispielsmenge $\{x_1,\ldots,x_{i+1}\} \setminus (T \cup \{x_{i+1}\})$ konsistent

ist. Wenn ja, füge die Menge $T \cup \{x_{i+1}\}$ zu $F_{\{x_1,\ldots,x_i.x_{i+1}\}}$ hinzu. Wenn nein, ist $T \cup \{x_{i+1}\}$ kein Element von $F_{\{x_1,\ldots,x_i,x_{i+1}\}}$.
Warum ist diese induktive Konstruktion korrekt? Falls $T \in F_{\{x_1,\ldots,x_i,x_{i+1}\}}$, dann gibt es ein Konzept $c \in \mathcal{C}$ mit $c \cap \{x_1,\ldots,x_i,x_{i+1}\} = T$. Wenn $x_{i+1} \notin T$, dann ist $T \in F_{\{x_1,\ldots,x_i\}}$. Wenn $x_{i+1} \in T$, dann ist $T \setminus \{x_{i+1}\} = \{x_1,\ldots,x_i\} \cap c \in F_{\{x_1,\ldots,x_i\}}$. Also gilt stets

$$T \cap \{x_1,\ldots,x_i,x_{i+1}\} \in F_{\{x_1,\ldots,x_i\}} \Rightarrow T \cap \{x_1,\ldots,x_i\} \in F_{\{x_1,\ldots,x_i\}}.$$

Für jedes i $(0 \leq i \leq s)$ besitzt die Menge $F_{\{x_1,\ldots,x_i\}}$ höchstens $\left(\frac{e \cdot i}{d}\right)^d = poly(i) = poly_1(s)$ viele Elemente. Somit kann $F_{\{x_1,\ldots,x_s\}}$ in Zeit $poly_2(s)$ berechnet werden. □

In Schritt 4 unseres Lernalgorithmus wird eine kurze Vereinigung mit dem gierigen Überdeckungsalgorithmus in Zeit $poly_3(s)$ berechnet. Damit läuft der Occam-Algorithmus für $V(\mathcal{C})$ in polynomieller Zeit bzgl. $\frac{1}{\varepsilon}, \frac{1}{\delta}, t$ und n. ■

3.2.9 Korollar a) Für jede konstante Dimension d gibt es einen polynomiellen PAC-Algorithmus, der

$$V(Rechteck_{n,d})$$

in Zeit $poly(n, t, \frac{1}{\varepsilon}, \frac{1}{\delta})$ lernt. (t ist hierbei die minimale Anzahl von Rechtecken im Zielkonzept).
b) Die Konzeptklasse $Konvexe\text{-}Polygone_n$ besitzt für jedes konvexe Polygon P in $\mathbb{R}^2$ die Menge $\{(x,y) \mid (x,y) \in \{-2^n, 2^n\}^2, (x,y)$ gehört zum Inneren von $P\}$ als Konzept. Die Konzeptklasse

$$Konvexe\text{-}Polygone_n$$

besitzt einen polynomiellen PAC-Algorithmus, der in Zeit $poly(n, t, \frac{1}{\varepsilon}, \frac{1}{\delta})$ lernt. (t ist hierbei die minimale Anzahl von Eckpunkten im Zielkonzept).

Beweis a) und b) folgen direkt aus Satz 3.2.6: Es ist $VC(Rechteck_{n,d}) = 2d$ und $VC(Threshold_{2,n}) = 3$. ($Threshold_{2,n}$ besitzt für jede Hyperebene $ax + by = c$ im $\mathbb{R}^2$ die Menge $\{(x,y) \mid (x,y) \in \{-2^n, 2^n\}^2, ax + by \geq c\}$ als Konzept.) Weiterhin sind beide Konsiztenzprobleme in polynomieller Zeit lösbar.
Beachte für b), daß $Konvexe\text{-}Polygone_n = D(Threshold_{2,n})$ ist. □

3.3 Zusammenfassung

Polynomielle Lernbarkeit (im PAC-Sinn) folgt für (parametrisierte) Konzeptklassen stets dann, wenn das Konsistenzproblem effizient lösbar ist. Damit folgt unmittelbar die polynomielle Lernbarkeit von Konzeptklassen wie $k\text{-}CNF$, $k\text{-}DNF$, $Rechteck_{n,d}$ und Sym_n.

Wir haben gezeigt, daß die Existenz eines polynomiellen PAC-Algorithmus die Existenz eines effizienten probabilistischen Algorithmus bedingt, der das Konsistenzproblem für die Konzeptklasse löst. Deshalb konnten wir - unter der Annahme $R \neq NP$ - zeigen, daß es keine polynomiellen PAC-Algorithmen gibt, die k-Term DNF durch k-Term DNF erlernen. In den Übungen betrachten wir ein zweiten Beispiel: Es wird gezeigt, daß (sich nicht überschneidende) Manhattan-Kurvenzüge im $\mathbb{R}^2$ nicht polynomiell lernbar sind, wenn wir Manhattan-Kurvenzüge auch als Hypothesenklasse verwenden.

Damit erhalten wir negative Ergebnisse für **repräsentations-abhängiges** Lernen. Im nächsten Kapitel werden wir **repräsentations-unabhäginge** Aussagen über die Unmöglichkeit des polynomiellen Lernens einiger Konzeptklassen machen.

Wir haben uns weiterhin mit Occam-Algorithmen beschäftigt. Ein Occam-Algorithmus führt eine Art von Datenkomprimierung durch: Für s Beispiele ist ein Konzept mit Namenslänge $f(t) \cdot s^{\alpha}$ (für $0 \leq \alpha < 1$) zu konstruieren, wobei t die Namenslänge des zu lernenden Konzepts angibt. (Um zu einem effizienten Lernen zu führen, muß $f(t) = O(poly(t))$ gefordert werden.) Wir haben festgestellt, daß ein Occam-Algorithmus ein PAC-Algorithmus ist und haben die nötige Beispielsanzahl berechnet. Im Modell des PAC-Lernens ist „Occam's Razor“ also ein beweisbar richtiges Lernparadigma.

Der wesentliche Vorteil von Occam-Algorithmen ist, daß relativ wenige Beispiele ausreichen, wenn „kurze“ Konzepte zu lernen sind: Occam-Algorithmen passen sich also der Komplexität des Zielkonzeptes an.

Wir haben sodann den Begriff des Occam-Algorithmus verallgemeinert: Für ein vorgegebenes Komplexitätsmaß (gegeben durch die Parametrisierung der Konzeptklasse) ist eine kurze Hypothese zu finden. Die VC-Dimension der aufgestellten Hypothesen dient dabei als Definition der Länge. Wir haben dann die polynomielle Lernbarkeit von Vereinigungen oder Durchschnitten einer Konzeptklasse (mit endlicher VC-Dimension und polynomieller Lösung des Konsistenzproblems) etablieren können. Als eine Anwendung haben wir die polynomielle Lernbarkeit von konvexen Polygonen im $\mathbb{R}^2$ erhalten.

Der gierige Überdeckungsalgorithmus hat sich als eine wichtige Komponente im Entwurf einiger Occam-Algorithmen herausgestellt. Wir werden ein weiteres Beispiel im Kapitel 5 kennenlernen.

4 Kryptographie und PAC-Lernen

4.1 Das RSA Krypto-System

Wir beginnen mit einer Beschreibung der Fragestellung der public-key Kryptographie. Zwei Personen, Alice und Bob, möchten Nachrichten austauschen. Eine dritte Person, Eve, möchte die Nachrichten abfangen und, wenn möglich, entschlüsseln.

Das one-time pad stellt eine Möglichkeit dar, eine sichere Kommunikation zu gewährleisten. Alice und Bob treffen sich in einem stillen Kämmerlein und vereinbaren eine Liste L von zufällig gewählten Bits. Wenn Bob ein Bit b zu Alice senden möchte, wählt er das erste unbenutzte Bit c der gemeinsamen Liste L und sendet $b \oplus c$ zu Alice. Alice entschlüsselt

das erhaltene Bit $d = b \oplus c$, indem sie $d \oplus c$ berechnet.
Diese relativ sichere Methode hat den Nachteil, daß Alice und Bob sich treffen müssen; ein erheblicher Nachteil, wenn einer von beiden eine sichere Kommunikation mit vielen Personen benötigt.

Die public-key Kryptographie baut auf dem Begriff der Trapdoor-Funktion (Falltür-Funktion) auf. Eine Trapdoor-Funktion f_w ist effizient berechenbar, aber f_w^{-1} ist nur dann effizient berechenbar, wenn w bekannt ist. Mit Hilfe einer Trapdoor-Funktion f_w kommunizieren Alice und Bob nun wie folgt:

Alice stellt für sich eine Trapdoor-Funktion f_w auf und kommuniziert ein Programm zur Berechnung von f_w (über einen offenen Kanal). Bob kodiert seine Nachricht x mit dem öffentlichen Programm und sendet $f_w(x)$ (über den öffentlichen Kanal) zu Alice. Alice kann $f_w(x)$ effizient dekodieren, da sie w kennt. Eve kennt w nicht und kann deshalb keine effiziente Dekodierung erreichen.

Beispiel Die RSA Kodierung (**R**ivest, **S**hamir, **A**dleman, 1978):
Alice wählt 2 „riesige" Primzahlen p und q, sowie den Kodierungsexponenten e. Sie sendet $N = p \cdot q$ und e über einen öffentlichen Kanal. Bob berechnet $f_{p,q,e}(x) = x^e \bmod N$ und sendet $f_{p,q,e}(x)$. Hierbei muß gelten, daß $0 < x < N$ und $(x, N) = 1$.
[(a, b) bezeichnet den größten gemeinsamen Teiler der ganzen Zahlen a und b].
Alice nutzt ihr Wissen der Primfaktoren p und q, um $f_{p,q,e}(x)$ effizient zu dekodieren.

Wie werden zwei große Primzahlen p und q erzeugt?
Zur Erzeugung von p und q werden Zufallszahlen darauf getestet, ob sie Primzahlen sind. Da es (nach dem Primzahlsatz) genügend Primzahlen gibt, müssen im Durchschnitt nicht zu viele Versuche durchgeführt werden. Es gibt einen effizienten probabilistischen Primzahltest, der mit kleiner Wahrscheinlichkeit Zahlen die keine Primzahlen sind, als Primzahlen ausgibt. (Durch einen solchen Fehler wird die Kommunikation zwischen Alice und Bob gestört; Eve erhält jedoch keine zusätzliche Information.)

Wie wird der Kodierungsexponent e gewählt?
Alice berechnet den Wert der Euler-Funktion $\Phi(N)$; dabei ist $\Phi(N)$ die Anzahl aller $x \in \{1, \ldots, N-1\}$, die mit N teilerfremd sind. Für Primzahlen p und q mit $p \cdot q = N$ gilt:

$$\Phi(N) = p \cdot q - (p-1) - (q-1) - 1 = (p-1)(q-1),$$

denn nur die $p-1$ Vielfachen von q: $\{q, 2q, \ldots, (p-1)q\}$, die $q-1$ Vielfachen von p: $\{p, 2p, \ldots, (q-1)p\}$ und 0 sind nicht teilerfremd mit N.

Alice erzeugt sich eine Zufallszahl $e \in \{2, \ldots, \Phi(N)-1\}$, so daß e und $\Phi(N)$ teilerfremd sind.

Warum ist $x^e \bmod N$ effizient berechenbar?
Bob berechnet zuerst durch fortgesetztes Quadrieren $x \bmod N$, $x^2 \bmod N$, $x^4 \bmod N, \ldots,$ $x^{2^i} \bmod N, \ldots, x^{2^{\lfloor \log_2 N \rfloor}} \bmod N$ (insgesamt wird also $\lfloor \log_2 N \rfloor + 1$-mal quadriert). Danach berechnet Bob $x^e \bmod N$ durch Multiplikation der **geeigneten** Potenzen: Sei die Binärdar-

stellung von $e = \sum_{i=0}^{k} a_i \cdot 2^i$, mit $a_i \in \{0,1\}$, gegeben. Dann ist

$$x^e \equiv x^{\sum_{i=0}^{k} a_i \cdot 2^i} \equiv \prod_{i=0}^{k} x^{a_i \cdot 2^i} \equiv \prod_{i, a_i=1} x^{2^i} \bmod N;$$

$x^e \bmod N$ wird also als Produkt von höchstens $\lfloor \log_2 N \rfloor + 1$ Faktoren berechnet.

Warum kann Alice $x^e \bmod N$ effizient dekodieren?

Alice bestimmt zuerst den Dekodierungsexponenten d. d hat die Eigenschaft, daß

$$(x^e)^d \equiv x \bmod \mathrm{N}$$

gilt. Es ist also nur zu zeigen, daß d effizient bestimmbar ist, **wenn** die Primfaktoren p und q bekannt sind.

Der kleine Satz von Fermat besagt für eine Gruppe G mit Gruppenordnung $d = |G|$, daß $g^d = 1$ für alle $g \in G$ gilt. Wir wenden den kleinen Satz von Fermat auf die Gruppe Z_N^*, der für N primen (d.h. teilerfremden) Restklassen, mit der Operation Multiplikation modulo N an. Es ist also $Z_N^* = (\{x \mid 0 < x < N \text{ und } (x, N) = 1\}, \text{ Multiplikation } \bmod N)$.

Wie wir oben gesehen haben ist $\Phi(N) = (p-1)(q-1)$ die Gruppenordnung von Z_N^*. Für jedes x aus G gilt damit

$$x^{a \cdot \Phi(N)+r} \equiv x^{a \cdot \Phi(N)} x^r \equiv x^r \bmod N.$$

Alice bestimmt den Dekodierungsexponenten d so, daß

$$e \cdot d \equiv 1 \bmod \Phi(N)$$

gilt. Diese Gleichung ist lösbar, da e und $\Phi(N)$ teilerfremd sind. Es bleibt zu zeigen, daß Alice d auch effizient bestimmen kann.

Zuerst beachten wir, daß Alice $\Phi(N)$ bestimmen kann, da sie die Primfaktoren p und q kennt. Das Dekodierungsproblem ist damit auf die effiziente Berechnung multiplikativer Inverser zurückgeführt.

Wir zeigen jetzt allgemein, wie multiplikative Inverse berechnet werden: $a \cdot x \equiv 1 \bmod b$ sei zu lösen, wobei a und b teilerfremd sind. Wir werden x bestimmen, indem wir Euklid's Algorithmus zur Bestimmung des größten gemeinsamen Teilers von a und b aufrufen.

```
function Euclid(u, v)    /* u < v */
if u = 0 then
   return (v)
else
   return (Euclid(u mod v, u))
```

Der Euklid'sche Algorithmus berechnet (u, v) mit $O(\log_2(u+v))$ arithmetischen Operationen. Bezogen auf die Länge der Eingabe (also auf $\lceil \log_2(u+1) \rceil + \lceil \log_2(v+1) \rceil$) benötigt

der Euklid'sche Algorithmus somit lineare Rechenzeit.
(Zur Korrektheit: Die Teiler von u und v stimmen mit den Teilern von $v - \lfloor \frac{v}{u} \rfloor \cdot u \equiv v \bmod u$ und u überein.)

Analyse von *Euclid*(a, b): Wir rufen *Euclid*(a, b) auf und erhalten mit $u_0 = b$ und $u_1 = a$ die Folge

$$\begin{aligned} b = u_0 &= \alpha_0 \cdot u_1 + u_2 &&= \alpha_0 \cdot a + (a \bmod b) \\ u_1 &= \alpha_1 \cdot u_2 + u_3 && \\ &\vdots && \\ u_{s-2} &= \alpha_{s-2} \cdot u_{s-1} + u_s && \\ u_{s-1} &= \alpha_{s-1} u_s + u_{s+1}, && \text{mit } u_{s+1} = 0 \end{aligned}$$

von Zwischenergebnissen, mit $\alpha_i = \lfloor \frac{u_i}{u_{i+1}} \rfloor$. Insbesondere ist $u_s = (a, b) = 1$, und somit $u_{s-2} = \alpha_{s-2} \cdot u_{s-1} + 1$. Mit anderen Worten

$$1 = u_{s-2} - \alpha_{s-2} \cdot u_{s-1}.$$

Wir zeigen jetzt, daß die Folge der Zwischenergebnisse eine Darstellung der Form $1 = m_1 \cdot a + m_2 \cdot b$ ergibt; dann ist $1 \equiv m_1 \cdot a \bmod b$ und somit $\mathbf{m_1 \bmod b}$ das gesuchte multiplikative Inverse.

Wir wissen, daß $1 = u_{s-2} - \alpha_{s-2} \cdot u_{s-1}$. Andererseits gilt $u_{s-3} = \alpha_{s-3} \cdot u_{s-2} + u_{s-1}$ und damit $u_{s-1} = u_{s-3} - \alpha_{s-3} \cdot u_{s-2}$. Als Konsequenz erhalten wir:

$$1 = u_{s-2} - \alpha_{s-2} (u_{s-3} - \alpha_{s-3} \cdot u_{s-2}) = (1 + \alpha_{s-2} \cdot \alpha_{s-3}) u_{s-2} - \alpha_{s-2} \cdot u_{s-3}.$$

Durch wiederholtes Einsetzen erhalten wir eine Darstellung der Form $1 = m_1 \cdot a + m_2 \cdot b$.

Obwohl bisher nicht bekannt ist, ob die RSA-Kodierung (durch einen polynomiell Zeitbeschränkten Algorithmus) gebrochen werden kann, gibt es starke Anzeichen gegen die Existenz effizienter Dekodierungen:

- Die RSA-Kodierung ist seit 16 Jahren ungebrochen.
- Für die Wahl von $e = 2$ kann gezeigt werden, daß die effiziente Dekodierung von $x^2 \bmod N$ bedingt, daß die Primfaktoren von N durch einen effizienten Algorithmus gefunden werden können.
(Siehe M.O. Rabin, *Digital signatures and public key functions as intractable as factoring.*)

4.2 Konsequenzen für das PAC-Lernen

Wir kommen jetzt zum wesentlichen Punkt, nämlich dem **Zusammenhang zwischen PAC-Lernen und Kryptographie**: Um die von Bob gesandten Nachrichten zu entschlüsseln, kann Eve versuchen, die Dekodierung von Alice zu erlernen:

- Wenn Alice die Kodierungsfunktion f_w bekanntgibt, kann Eve zufällig Beispiele $(f_w(x), x)$ für die Dekodierungsfunktion f_w^{-1} generieren! (Diese Beispiele werden gemäß der Gleichverteilung erzeugt.)

- Eve wendet einen effizienten Lernalgorithmus an und erhält eine Hypothese, die fast mit f_w^{-1} übereinstimmt.

Wir haben noch nicht den Übergang zum Konzeptlernen erreicht. Hierzu benutzen wir ein Ergebnis von W. Alexi, B. Chor, O. Goldreich und C.P. Schnorr, *RSA and Rabin functions: certain parts are as hard as the whole*, SIAM Journal on Computing 17, pp. 194-209, 1988:

> Angenommen, ein probabilistischer polynomiell-zeitbeschränkter Algorithmus findet das niedrigst-wertige Bit von x (bei Eingabe $(e, N, x^e \bmod N)$) mit Wahrscheinlichkeit mindestens $\frac{1}{2} + \frac{1}{poly(\log_2 N)}$. Dann gibt es einen probabilistischen, polynomiell-zeitbeschränkten Algorithmus, der $x^e \bmod N$ (mit Wahrscheinlichkeit mindestens $\frac{3}{4}$) dekodiert.

Wir gehen jetzt zu einer parametrisierten Konzeptklasse für das RSA-System über:

4.2.1 Definition Seien p und q zwei Primzahlen und e ein Kodierungsexponent, mit $2 \leq e < (p-1)(q-1)$. Sei $N = p \cdot q$ und $r \equiv x^e \bmod N$. Die Konzeptklasse RSA_n besitzt für alle Tripel (p, q, e), mit $\lfloor \log_2 N \rfloor \leq n$, das Konzept $c_{p,q,e}$, wobei

$$c_{p,q,e} = \left\{ \left(N, e, r \bmod N, r^2 \bmod N, \ldots, r^{2^{\lfloor \log_2 N \rfloor}} \bmod N\right) \mid \begin{array}{l} \text{das niedrigst-wertige Bit} \\ \text{von } x \text{ ist } 1 \end{array} \right\}$$

gilt. Die parametrisierte Konzeptklasse RSA ist definiert durch $RSA = (RSA_n \mid n \in \mathbb{N})$.

[Alle Konzepte mit annähernd gleicher Eingabelänge werden zusammengefaßt: Ein Konzept aus RSA_n enthält $\lfloor \log_2 N \rfloor + 2 \leq n+2$ Komponenten mit jeweils höchstens $\lfloor \log_2 N \rfloor + 1 \leq n+1$ Bits. Die Eingabelänge ist also durch $O(n^2)$ beschränkt.]

Mit dem nächsten Satz zeigen wir, daß jeder effiziente PAC-Algorithmus für genügend schwierige Konzeptklassen (die die Konzeptklasse RSA_n enthalten) das RSA-System „knackt“. (Somit wird es vermutlich keine effizienten PAC-Algorithmen für derart mächtige Konzeptklassen geben.)

4.2.2 Satz *Sei eine parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$, mit $n = \lfloor \log_2 N \rfloor$ gegeben. Sei $poly(n)$ ein Polynom, so daß $RSA_n \subseteq \mathcal{C}_{poly(n)}$ (für alle n) gilt. Wenn es einen polynomiell-zeitbeschränkten PAC-Algorithmus für $\mathcal{C}$ gibt, dann gibt es einen probabilistischen, polynomiell-zeitbeschränkten Algorithmus, der bei Eingabe $(e, N, x^e \bmod N)$ die Ausgabe x (mit Wahrscheinlichkeit mindestens $\frac{3}{4}$) berechnet.*

Beweis Sei L ein polynomiell zeitbeschränkter PAC-Algorithmus für $\mathcal{C}$. Ein effizienter probabilistischer Algorithmus, der das RSA-System bricht, sieht wie folgt aus:

1. Es werden positive und negative Beispiele gemäß der Gleichverteilung generiert. (Wie? Sei y, mit $0 < y < N$ zufällig gewählt, wobei y und N teilerfremd seien. Berechne $y^e \bmod N$ und gib $(N, e, y^e \bmod N, \ldots, (y^e)^{2^{\lfloor \log_2 N \rfloor}} \bmod N, y_0)$ als durch y_0 klassifiziertes Beispiel aus, wobei y_0 das niedrigst-wertige Bit von y ist.)

2. Rufe L mit den Parametern $\varepsilon = \frac{1}{2} - \frac{1}{poly(\log_2 N)}$ und $\delta = \frac{1}{2 \cdot poly(\log_2 N)}$ für obige Beispiele auf.

3. L gibt eine Hypothese aus. (Diese Hypothese ist irgendein Programm, das das letzte Bit von x bei Eingabe $(N, e, x^e \bmod N, \ldots, (x^e)^{2^{\lfloor \log_2 N \rfloor}})$ berechnet.)
   Die Fehlerwahrscheinlichkeit der Hypothese ist höchstens

   $$\underbrace{\frac{1}{2 \cdot poly(\log_2 N)}}_{\text{Wahrscheinlichkeit, daß Fehler zu groß ist}} + \underbrace{\frac{1}{2} - \frac{1}{poly(\log_2 N)}}_{\text{maximaler Fehler } \varepsilon} = \frac{1}{2} - \frac{1}{2 \cdot poly(\log_2 N)}.$$

   Somit ist die Wahrscheinlichkeit, daß x_0 korrekt rekonstruiert wird, mindestens

   $$1 - \left(\frac{1}{2} - \frac{1}{2 \cdot poly(\log_2 N)}\right) = \frac{1}{2} + \frac{1}{2 \cdot poly(\log_2 N)}$$

   und das Ergebnis von Alexi *et al.* kann angewendet werden.

4. Nach dem Ergebnis von Alexi *et al.* kann ein Algorithmus konstruiert werden, der x (mit Wahrscheinlichkeit mindestens $\frac{3}{4}$) dekodiert.

□

Wir haben im Beweis von Satz 4.2.2 nur PAC-Algorithmen verwendet, die einen Fehler nahe bei $\frac{1}{2}$ erreichen. Solche Lernalgorithmen beschreibt die folgende Definition:

4.2.3 Definition Ein Lernalgorithmus L für eine parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ heißt genau dann <u>schwacher PAC-Algorithmus</u>, wenn es ein Polynom $poly(n)$ gibt, so daß für jedes δ, für jedes n, für jede Verteilung D und für jedes Konzepte $c \in \mathcal{C}_n$

$$prob\left[Fehler_D(c, h_c) \leq \frac{1}{2} - \frac{1}{poly(n)}\right] \geq 1 - \delta$$

gilt, wobei h_c die von L gefundene Hypothese ist.

Bemerkung Der Beweis von Satz 4.2.2 zeigt, daß der Satz auch für schwache PAC-Algorithmen gilt, also für PAC-Algorithmen, die nur unwesentlich bessere Ergebnisse als zufällige Entscheidungen liefern. Der Beweis von Satz 4.2.2 zeigt sogar, daß erfolgreiches Lernen gemäß der Gleichverteilung zum Brechen der RSA-Kodierung führt.

Welche „vernünftigen“ Konzeptklassen sind mächtig genug, um RSA zu enthalten und besitzen somit (unter der Annahme, daß das RSA-System nicht gebrochen werden kann) keine effizienten PAC-Algorithmen?

4.2.4 Satz *Angenommen, es gibt keinen probabilistischen, polynomiell-zeitbeschränkten Algorithmus, der bei Eingabe $(e, N, x^e \bmod N)$ (wobei N das Produkt zweier Primzahlen ist) die Ausgabe x mit Wahrscheinlichkeit mindestens $\frac{3}{4}$ berechnet. Dann gibt es keine polynomiellen schwachen PAC-Algorithmen für die folgenden parametrisierten Konzeptklassen:*

a) $Schaltkreis_n$,
(Die Konzeptklasse $Schaltkreis_n$ besitzt für jeden $\wedge, \vee, \neg$-Schaltkreis S mit

- *Variablenmenge $\{x_1, \ldots, x_n\}$,*
- *Fan-in[3] höchstens 2,*
- *genau einer Ausgabe und*
- *Tiefe höchstens $\log_2 n$*

ein Konzept $wahr(S)$. Das Konzept besteht aus allen von S akzeptierten Eingaben: $wahr(S) = \{x \mid x \in \{0,1\}^n, S(x) = 1\}$.)

b) $Formel_n$,
($Formel_n$ besitzt für jede aussagenlogische Formel α mit $\{x_1, \ldots, x_n\}$ als Variablenmenge und Formellänge[4] höchstens n ein Konzept $wahr(\alpha)$. Das Konzept besteht aus allen Eingaben, die α erfüllen: $wahr(\alpha) = \{x \mid x \in \{0,1\}^n, \alpha(x) = 1\}$.)

c) $Threshold_{d,n}$, für eine geeignete Konstante d
($Threshold_{d,n}$ besitzt für jeden Threshold-Schaltkreis T mit $\{x_1, \ldots, x_n\}$ als Variablenmenge und Tiefe höchstens d ein Konzept $wahr(T)$. Es ist $wahr(T) = \{x \mid x \in \{0,1\}^n, T(x) = 1\}$.)

Beweis a) Mit Satz 4.2.2 reicht es zu zeigen, daß $RSA_n \subseteq Schaltkreis_{poly(n)}$ gilt. Also müssen wir für jedes Konzept $c_{p,q,e} \in RSA_n$ einen Schaltkreis S konstruieren, der genau die Menge $c_{p,q,e}$ akzeptiert.

Die Elemente aus $c_{p,q,e}$ haben die Form $w = (N, e, r \bmod N, r^2 \bmod N, \ldots, r^{2^{\lfloor \log_2 N \rfloor}})$, mit $N = p \cdot q$, $r \equiv x^e \bmod N$, wobei das niedrigst-wertige Bit von x den Wert 1 hat. Die

[3]Anzahl der Eingänge in einen Baustein
[4]Anzahl der logischen Operatoren $\wedge, \vee, \neg$ in α

Eingabelänge von S ist somit $(n+1)(n+2)$, mit $n \geq \lfloor log_2 N \rfloor$. S soll eine Eingabe v der Länge $(n+1)(n+2)$ genau dann akzeptieren, falls $v \in RSA_n$ ist. Der zu konstruierende Schaltkreis wird eine Tiefe von $O(\log n)$ besitzen; wir fügen polynomiell viele Eingabebits hinzu, um die Tiefe auf strikt logarithmisch (in der Anzahl aller Eingabebits) zu senken.
<u>Konstruktion von S:</u> S überprüft, ob seine Eingabe eine Binärkodierung der Form (N, e, $r \bmod N, r^2 \bmod N, \ldots, r^{2^{\lfloor log N \rfloor}} \bmod N$) mit $r \equiv (x^e \bmod N)$ ist. (Einfaches Quadrieren ist für einen Schaltkreis in logarithmischer Tiefe möglich. Da alle Potenzen von $r \bmod N$ als Eingabe vorliegen, braucht der Schaltkreis nur zu überprüfen, ob $(r^{2^i})^2 \equiv r^{2^{i+1}} \bmod N$ gilt. Dies wird *parallel* $\lfloor \log_2 N \rfloor$ mal in Tiefe $O(\log\log_2 N)$ durchgeführt, bzw. höchstens n mal in Tiefe $O(\log n)$.)
Wir kodieren den Dekodierungsexponenten d (mit $(x^e)^d \equiv x \bmod N$ für alle $(x, N) = 1$) direkt „in den Schaltkreis S“ hinein. Dann berechnet S den Wert $x \equiv r^d \bmod N$ und gibt das niedrigst-wertige Bit von x aus.
Das Problem ist somit die Berechnung von $r^d \bmod N$. Hierzu benutzen wir die bereits als Eingaben zur Verfügung gestellten Potenzen $r^{2^i} \bmod N$: Nach Konstruktion ist die Binärdarstellung von $d = \sum_{i=0}^{k} a_i \cdot 2^i$, mit $a_i \in \{0,1\}$, dem Schaltkreis S bekannt. Es gilt

$$x \equiv r^d \equiv r^{\sum_{i=0}^{k} a_i \cdot 2^i} \equiv r^{\sum_{i,a_i=1} 2^i} \equiv \prod_{i,a_i=1} r^{2^i} \bmod N,$$

und $r^d \bmod N$ ist somit ein Produkt von höchstens $\lfloor log_2 N \rfloor + 1 \leq n+1$ Potenzen. Auch das Wissen welche a_i den Wert 1 haben, d.h. welche Potenzen zu wählen sind, „programmieren“ wir direkt in den Schaltkreis S hinein.
Wir müssen somit einen Schaltkreis S finden, der höchstens $\lfloor \log_2 N \rfloor + 1 \leq n+1$ viele Zahlen mit $\lceil \log_2 N \rceil \leq n+1$ Bits in Tiefe $O(\log n)$ miteinander multipliziert.
Die Existenz eines solchen Schaltkreises folgt aus der Arbeit von P.W. Beame, S.A. Cook und H.J. Hoover, *Log depth circuits for division and related problems*, SIAM Journal on Computing 15, pp. 994-1003, 1986.
b) Für jeden Schaltkreis S der Tiefe t gibt es eine aussagenlogische Formel α_S mit $\alpha_S(x) = 1 \Leftrightarrow S(x) = 1$, für alle $x \in \{0,1\}^n$. Die Formel α_S hat Formellänge höchstens $2^t - 1$.
Diese Aussage, sowie der Rest des Arguments, ist Thema von Aufgabe 8.2 b).
c) Wird hier nicht bewiesen. □

Die drei behandelten Konzeptklassen sind extrem schwierig, und die Nicht-Existenz von effizienten PAC-Algorithmen ist nicht zu überraschend. Das folgende negative Ergebnis für endliche Automaten ist sehr enttäuschend:

4.2.5 Satz *Angenommen, es gibt keinen probabilistischen, polynomiell-zeitbeschränkten Algorithmus, der bei Eingabe $(e, N, x^e \bmod N)$ (wobei N das Produkt zweier Primzahlen ist) die Ausgabe x mit Wahrscheinlichkeit mindestens $\frac{3}{4}$ berechnet. Dann gibt es keine polynomiellen schwachen PAC-Algorithmen für die parametrisierte Konzeptklasse $Aut_{n,\{0,1\}}$.*

Beweis Wir beschreiben die Beweisidee; die Details werden im Übungsblatt 8 behandelt. Unser Ziel ist eine *Reduktion* von $Schaltkreis_n$ auf $Aut_{poly(n),\{0,1\}}$. Dazu klären wir zuerst, wie ein Schaltkreis S durch einen endlichen Automaten A_S mit wenigen Zuständen ausgewertet werden kann. Mit anderen Worten, wir müssen zuerst eine speichereffiziente Auswertung eines Schaltkreises besprechen. Dies gelingt mit Hilfe des *Pebble Spiels*:

Regeln des Pebble-Spiels:

- Ein Pebble darf stets auf eine Quelle des Schaltkreises gesetzt werden. (Das entsprechende Eingabebit wird in ein Register geladen.)
- Ein Pebble darf stets gelöscht werden. (Ein Register darf stets gelöscht werden.)
- Wenn alle Vorgänger eines Bausteins Pebbles besitzen, dann darf der Baustein selbst gepebbelt werden. (Wenn alle notwendigen Zwischenergebnisse gespeichert sind, dann kann der Baustein berechnet und das Ergebnis in ein Register geladen werden.)

Das Ziel des Pebble-Spiels ist es, den Ausgabebaustein des Schaltkreises zu pebbeln. Dabei soll sowohl die Anzahl der benutzten Pebbles (berechnet als das Maximum über alle Züge des Pebble-Spiels) als auch die Anzahl der Züge so klein wie möglich sein. Für Schaltkreise kleiner Tiefe kann das Pebble-Spiel effizent gespielt werden:

4.2.6 Lemma Sei S ein $\wedge, \vee, \neg$-Schaltkreis mit Fan-in höchstens 2 und genau einer Ausgabe. Die Tiefe von S sei t. Dann kann S mit höchstens $t+2$ Pebbles in Zeit höchstens $2^{t+1}-1$ gepebbelt werden.

Beweis Aufgabe 8.3

Wir konstruieren jetzt einen endlichen Automaten A_S, der ein optimales Pebble-Spiel für S Zug-um-Zug simuliert. Um diese Simulation zu ermöglichen, wiederholen wir die Eingabe oft genug:

4.2.7 Lemma Sei S ein $\wedge, \vee, \neg$-Schaltkreis mit Fan-in höchstens 2 und genau einer Ausgabe. Für jeden Schaltkreis S mit n Eingaben und Tiefe t (wobei $\lfloor \log_2 n \rfloor \leq t$) gibt es einen endlichen Automaten A_S mit höchstens $poly(2^t)$ Zuständen, so daß für alle $w \in \{0,1\}^n$ gilt:

$$A_S \text{ akzeptiert } \underbrace{w\ w\ \ldots\ w}_{2^{t+1}-1 \text{ mal}} \Leftrightarrow S \text{ akzeptiert } w.$$

Beweis Aufgabe 8.4

Wir können jetzt den Beweis des Satzes vervollständigen: Wir wissen, daß Schaltkreise mit logarithmischer Tiefe keine effizienten PAC-Algorithmen besitzen, selbst wenn die

Beispiele gemäß Gleichverteilung generiert werden. Angenommen, es gibt einen polynomiellen PAC-Algorithmus L für $Aut_{poly(n),\{0,1\}}$. Sei D_n die Gleichverteilung auf allen Worten der Form

$$\underbrace{w\ w\ \dots\ w}_{2^{t+1}-1 \text{ mal}},$$

für $w \in \{0,1\}^n$. L lernt $Aut_{poly(n),\{0,1\}}$ ebenfalls mit der Verteilung D_n effizient. Lemma 4.2.7 besagt, daß wir dann auch $Schaltkreis_n$ mit der Gleichverteilung effizient lernen können! Wir erhalten den gewünschten WIDERSPRUCH, da ein effizienter PAC-Algorithmus für $Aut_{n,\{0,1\}}$ einen effizienten PAC-Algorithmus für $Aut_{poly(n),\{0.1\}}$ bedingt. $\square$

4.3 Zusammenfassung

Dieses Kapitel hat gezeigt, wann eine Konzeptklasse $\mathcal{C}$ schwer zu lernen ist: Nämlich dann, wenn die Konzepte aus RSA_n auch in $\mathcal{C}_{poly(n)}$ enthalten sind. Insbesondere kann ein Berechnungsmodell (wie das Modell der Schaltkreise logarithmischer Tiefe) dann nicht gelernt werden, wenn in dem Berechnungsmodell n Zahlen von je n Bits multipliziert werden können. *Große Berechnungsskraft impliziert schwieriges oder unmögliches Lernen.*

Wie deprimierend sind die obigen Ergebnisse? Schaltkreise logarithmischer Tiefe, Formeln polynomieller Länge und Threshold-Schaltkreise konstanter Tiefe sind mächtige Berechnungsmodelle. Es ist nicht zu verwunderlich, daß „anständige“ Lernmodelle Schwierigkeiten haben, die entsprechenden Konzeptklassen zu erlernen.

Das negative Ergebnis für endliche Automaten ist jedoch enttäuschend. Welche Möglichkeiten bleiben uns:

1. Die Konstruktion schwacher PAC-Algorithmen,
2. die Einschränkung auf spezielle Verteilungen und
3. eine aktive Teilnahme des Lerners.

Die erste Möglichkeit müssen wir verwerfen: Die negativen Ergebnisse gelten auch für schwache PAC-Algorithmen!

Die Einschränkung auf spezielle Verteilungen, wie die Gleichverteilung, hilft nicht für die komplexen Konzeptklassen. Wir werden aber im nächsten Kapitel sehen, daß das Wissen der Verteilung von Vorteil sein kann.

Im Kapitel 6 behandeln wir ein aktiveres Lernmodell und werden effiziente Lernalgorithmen kennenlernen, die endliche Automaten **exakt** rekonstruieren.
Aber auch in aktiveren Lernmodellen können immer noch negative Ergebnisse gezeigt werden (z. B. für Schaltkreise logarithmischer Tiefe, Formeln polynomieller Länge und Threshold-Schaltkreise konstanter Tiefe). Siehe dazu D. Angluin und M. Kharitonov,

When won't membership queries help?, Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, pp. 444-454, 1991. Wie bereist erwähnt: Die obigen drei Konzeptklassen sind zu komplex, um in „anständigen" Lernmodellen effizent gelernt werden zu können.

Das Ziel einer Theorie des algorithmischen Lernens ist eine Katalogisierung in effizient lernbare und nicht effizient lernbare Konzeptklassen. Wir haben hier einen ersten Schritt in diese Richtung gemacht. Die Lernkomplexität wichtiger Konzeptklassen, wie DNF-Formeln, endliche Automaten und Entscheidungsbäume, haben wir bisher offengelassen: Diese Konzeptklassen werden in den nächsten Kapiteln behandelt.

5 Verteilungsspezifisches Lernen

Wir haben in Kapitel 4 gesehen, daß sogar die Beschränkung auf die Gleichverteilung für schwierige Lernprobleme (wie das Lernen von Schaltkreisen logarithmischer Tiefe) **nicht** zu effizientem Lernen führt.
Wir werden in diesem Kapitel aber ein Beispiel kennenlernen bei dem eine Einschränkung auf die Gleichverteilung zu einem „fast-effizienten" Lernen führt: Das Lernen von DNF-Formeln.

5.1 Das Lernen von DNF-Formeln

Im folgenden sei $\alpha \equiv \bigvee_{i=1}^{t} m_i$ die zu lernende DNF_n-Formel, wobei m_i ein Monom über der Variablenmenge $\{x_1, \ldots, x_n\}$ ist. Wir nennen eine Eingabe $x \in \{0,1\}^n$ positiv (bzw. negativ), wenn $\alpha(x) = 1$ (bzw. $\alpha(x) = 0$).

Es sind zwei Modelle der Gleichverteilung möglich: Im ersten Modell werden die Beispiele durch die Gleichverteilung D produziert. Im zweiten Modell werden die positiven (negativen) Beispiele durch die Gleichverteilung D_+ (bzw. D_-) mit Wahrscheinlichkeit $\frac{1}{pos}$ (bzw. $\frac{1}{neg}$) generiert, wobei pos $=|\{x \mid x$ ist ein positives Beispiel $\}|$ und neg $=2^n-$pos ist.
Falls es nur „sehr wenige" positive Beispiele gibt, generiert das erste Modell mit „großer" Wahrscheinlichkeit keine positive Beispiele. Der Lernalgorithmus wird dann die leere DNF-Formel als Hypothese ausgeben. Das zweite Modell hingegen generiert dieselbe Anzahl von positiven und negativen Beispielen: die leere DNF-Formel wird keine Lösung sein. Wir beschränken uns auf das realistischere zweite Modell.

Der Ansatz des zu entwerfenden Lernalgorithmus kann am besten durch die beiden folgenden Ziele erklärt werden:
1. Ziel: Wir möchten zeigen, daß es eine DNF_n-Formel α^* gibt, die α gut **approximiert** und aus höchstens t **kurzen** Monomen besteht.
2. Ziel: Der Lernalgorithmus wird deshalb versuchen, eine approximierende Disjunktion kurzer Monome für α zu finden.

1. Ziel: Wie können wir die Monome für eine Formel α^* kurz halten und α gut approximieren?
Wir unterscheiden hierbei zwei Situationen:

- Die Formel α bildet mehr als die Hälfte aller Eingaben auf 1 ab: Wenn t relativ klein ist (zum Beispiel polynomiell in n), dann müssen die meisten Monome kurz sein. Lange Monome aus α sind unwichtig, da sie nur wenige der 2^n Eingaben auf 1 abbilden. Diese langen Monome „werfen wir" für α^* raus.
- Die Formel α bildet weit weniger als die Hälfte aller Eingaben auf 1 ab: Dann besteht α aus langen Monomen. Für α^* stellen wir kurze Monome zur Verfügung, indem wir jedes lange Monom aus α durch ein kurzes Teilmonom ersetzen. Nun werden mehr Eingaben als zuvor akzeptiert. Wir müssen also aufpassen, daß der Fehler (nämlich das Akzeptieren negativer Eingaben) nicht zu groß wird.

Wir führen das folgende zentrale Konzept ein:

5.1.1 Definition Sei γ eine reelle Zahl aus $]0,1]$. Ein Monom m mit $m \Rightarrow \alpha$ heißt genau dann $\underline{\gamma\text{-unwichtig für } \alpha}$, wenn

$$prob[m(x) = 1 \mid \alpha(x) = 1] \leq \gamma$$

gilt. m heißt $\underline{\gamma\text{-wichtig für } \alpha}$, wenn

$$prob[m(x) = 1 \mid \alpha(x) = 1] > \gamma$$

gilt.

Mit anderen Worten: Ein Monom ist nur dann wichtig, wenn es viele der positiven Eingaben „beisteuert".

Wir werden die folgende Abschätzung benutzen, um die Unwichtigkeit eines Monoms m_i zu etablieren:

$$\begin{aligned} prob[m_i(x) = 1 \mid \alpha(x) = 1] &= \frac{prob[m_i(x) = 1 \text{ und } \alpha(x) = 1]}{prob[\alpha(x) = 1]} \\ &\leq \frac{prob[m_i(x) = 1]}{prob[\alpha(x) = 1]}. \end{aligned}$$

(Wir haben benutzt, daß die Wahrscheinlichkeit „prob" gemäß der Gleichverteilung definiert ist.) Mit $|m_i|$ bezeichnen wir die Anzahl der Literale in m_i. Dann gilt

$$prob[m_i = 1] = 2^{-|m_i|},$$

denn: Für ein Monom mit k Literalen müssen die k (den Literalen entsprechenden) Bits festgelegt werden. Somit gilt:

$$prob[m_i(x) = 1 \mid \alpha(x) = 1] \leq \frac{2^{-|m_i|}}{prob[\alpha(x) = 1]}. \qquad (*)$$

Wir setzen $\gamma = \frac{\varepsilon}{t}$ und entfernen alle Monome m_i aus α, die γ-unwichtig sind. Somit erhalten wir eine DNF_n-Formel α^*, die nur aus γ-wichtigen Monomen besteht. Wie hoch ist dann der Approximationsfehler? Ein Fehler kann nur auf positiven Eingaben enstehen, die durch Auslassen von Monomen als negativ klassifiziert werden. Bei jedem entfernten Monom kann ein Fehler von höchstens $\frac{\varepsilon}{t}$ auftreten. Da höchstens t (also alle) Monome entfernt werden, gilt:

$$\begin{aligned} prob[\alpha(x) \neq \alpha^*(x)] &= prob[\alpha(x) = 1 \text{ und } \alpha^*(x) = 0] \\ &\leq \tfrac{\varepsilon}{t} \cdot t = \varepsilon. \end{aligned}$$

Zunächst beschäftigen wir uns mit der **ersten Situation**:
Angenommen, es gilt $prob[\alpha(x) = 1] \geq \frac{1}{2}$. Dann gilt wegen $(*)$:

$$\begin{aligned} prob[m_i(x) = 1 \mid \alpha(x) = 1] &\leq 2 \cdot prob[m_i(x) = 1] \\ &= 2 \cdot 2^{-|m_i|}. \end{aligned}$$

Setze $\alpha^* \equiv \bigvee_{i, m_i \text{ ist } \frac{\varepsilon}{t}\text{-wichtig}} m_i$, d.h. wir entfernen alle $\frac{\varepsilon}{t}$-unwichtigen Monome.

- Der Approximationsfehler ist (wie wir oben überlegt haben) höchstens ε.
- Alle Monome aus α^* (dies sind alle $\frac{\varepsilon}{t}$-wichtigen Monome für α) besitzen höchstens $\log_2(\frac{2t}{\varepsilon})$ Literale.
  (Warum? Alle Monome mit mindestens $\log_2(\frac{2t}{\varepsilon})$ Literalen sind $\frac{\varepsilon}{t}$-unwichtig, denn

$$\begin{aligned} & 2 \cdot 2^{-|m_i|} \leq \frac{\varepsilon}{t} \\ \Leftrightarrow\quad & 2^{-|m_i|} \leq \frac{\varepsilon}{2t} \\ \Leftrightarrow\quad & -|m_i| \leq \log_2(\tfrac{\varepsilon}{2t}) \\ \Leftrightarrow\quad & |m_i| \geq \log_2(\tfrac{2t}{\varepsilon}). \quad) \end{aligned}$$

Wir haben also für die erste Situation eine Formel α^* gefunden, die α mit höchstens t kurzen Monomen gut approximiert.

Wie gehen wir in der **zweiten Situation** vor?
Es ist nun $prob[\alpha(x) = 1]$ klein und somit α eine Disjunktion von recht langen Monomen. Das Entfernen der langen Monome kann somit alle positiven Eingaben zerstören. Stattdessen wählen wir für jedes lange Monom m_i von α ein kurzes Teilmonom für α^*. Die Gefahr hierbei ist, daß ein großer Fehler auf den negativen Eingaben entsteht: Negative Eingaben können von α^* akzeptiert werden. Daß der Fehler nicht zu groß wird, besagt das folgende Lemma:

5.1.2 Lemma Es sei $\alpha \equiv \bigvee_{i=1}^{t} m_i$. Für jedes $k \in \mathbb{N}$ und jedes $i \in \{1, \ldots, t\}$ gilt:

- m_i ist 2^{-k}-unwichtig
- **oder** jedes Teilmonom m_i' von m_i hat die Eigenschaft:

$$prob[m_i'(x) = 1 \mid \alpha(x) = 0] \leq 2^{-k},$$

  falls m_i' genau $\min\{k, |m_i|\}$ Literale besitzt.

(**Entweder** ist ein Monom m_i unwichtig, und wir können es entfernen, **oder** wir können es durch ein kurzes Teilmonom mit kleinem Fehler ersetzen.)

Beweis Sei m_i ein beliebiges Monom aus α.
Fall 1: m_i hat höchstens k Literale.
Dann ist $m_i' = m_i$, und es wird kein Fehler gemacht:

$$prob[m_i(x) = 1 \mid \alpha(x) = 0] = 0.$$

Fall 2: m_i hat mehr als k Literale.
Wir setzen

$$\begin{aligned} richtig &= \{x \mid x \in \{0,1\}^n, \alpha(x) = 1\}, \\ falsch &= \{x \mid x \in \{0,1\}^n, \alpha(x) = 0\}, \\ wahr_i &= \{x \mid x \in \{0,1\}^n, m_i(x) = 1\}, \\ rest_i &= \{x \mid x \in \{0,1\}^n, \alpha(x) = 1 \text{ und } m_i(x) = 0\}. \end{aligned}$$

Dann ist

$$|wahr_i| + |falsch| + |rest_i| = 2^n$$

und

$$|richtig| = |wahr_i| + |rest_i|,$$

denn $richtig = wahr_i \,\dot\cup\, rest_i$.

Fall 2.1: $|rest_i| > (2^k - 1)(2^{n-|m_i|})$,
d.h. die Menge $rest_i$ ist groß; somit trägt das Monom m_i wenig zu α bei und „sollte“ unwichtig sein. Wir werden zeigen, daß m_i tatsächlich 2^{-k}-unwichtig ist, womit der erste Fall des Lemmas eintritt. Zuerst beachte, daß $|wahr_i| + |rest_i|$ die Anzahl der positiven

Eingaben ist. Deshalb gilt

$$\begin{aligned}
prob[m_i(x) = 1 \mid \alpha(x) = 1] &\leq \frac{2^{-|m_i|}}{prob[\alpha(x) = 1]} \\
&= \frac{2^{-|m_i|}}{\frac{|richtig|}{2^n}} \\
&= \frac{2^{n-|m_i|}}{|wahr_i| + |rest_i|} \\
&< \frac{2^{n-|m_i|}}{2^{n-|m_i|} + (2^k - 1)(2^{n-|m_i|})} \\
&= \frac{1}{1 + 2^k - 1} = 2^{-k},
\end{aligned}$$

d.h. m_i ist 2^{-k}-unwichtig.

Fall 2.2: $|rest_i| \leq (2^k - 1)(2^{n-|m_i|})$, d.h. die Menge $rest_i$ ist klein; somit trägt das Monom m_i viel zu α bei und wir können im allgemeinen nicht erwarten, daß m_i unwichtig ist. Wir müssen also verifizieren, daß der zweite Fall des Lemmas eintritt: Sei m_i' ein Teilmonom von m_i mit k Literalten. Wenn $m_i(x) = 1$ ist, dann ist auch $m_i'(x) = 1$ und $\alpha(x) = 1$. Also werden die $2^{n-|m_i|}$ von m_i akzeptierten Eingaben von m_i' ebenfalls (und richtigerweise) akzeptiert. Dann wird m_i' von höchstens $2^{n-k} - 2^{n-|m_i|}$ negativen Eingaben erfüllt. Es gilt:

$$\begin{aligned}
prob[m_i'(x) = 1 \mid \alpha(x) = 0] &= \frac{prob[m_i'(x) = 1 \text{ und } \alpha(x) = 0]}{prob[\alpha(x) = 0]} \\
&\leq \frac{2^{n-k} - 2^{n-|m_i|}}{|falsch|} \\
&= \frac{2^{n-k} - 2^{n-|m_i|}}{2^n - |wahr_i| - |rest_i|} \\
&\leq \frac{2^{n-k} - 2^{n-|m_i|}}{2^n - 2^{n-|m_i|} - (2^k - 1)(2^{n-|m_i|})} \\
&= \frac{2^{n-k} - 2^{n-|m_i|}}{2^n - 2^k \cdot 2^{n-|m_i|}} \\
&= \frac{2^{n-k} - 2^{n-|m_i|}}{2^k(2^{n-k} - 2^{n-|m_i|})} \\
&= 2^{-k},
\end{aligned}$$

und der zweite Fall tritt, wie gefordert, ein. □

Für $\alpha \equiv \bigvee_{i=1}^{t} m_i$ entferne alle Monome m_i, die $\frac{\varepsilon}{t}$-unwichtig für α sind. (Der Fehler auf den positiven Eingaben übersteigt ε nicht. Ein Fehler auf den negativen Eingaben entsteht nicht.) Für alle übrigen Monome m_i wähle ein beliebiges Teilmonom mit $\lceil \log_2 \frac{t}{\varepsilon} \rceil$ Literalen, und bezeichne die Disjunktion der entsprechenden kurzen Monome mit α^*. (Kein weiterer Fehler auf den positiven Eingaben entsteht. Gemäß Lemma 5.1.2 wird ein Fehler von höchstens $2^{-\lceil \log_2 \frac{t}{\varepsilon} \rceil} \leq \frac{\varepsilon}{t}$ pro Monom nicht überschritten. Insgesamt ist der Fehler auf den negativen Eingaben somit durch höchstens $t \cdot \frac{\varepsilon}{t} = \varepsilon$ beschränkt.) Dann hat α^*

- höchstens t Monome mit höchstens $\lceil \log_2 \frac{t}{\varepsilon} \rceil$ Literalen,
- und es gilt $prob[\alpha(x) \neq \alpha^*(x)] \leq \varepsilon$.

Unser 1. Ziel haben wir erreicht: Wir wissen, daß es stets eine kurze Disjunktion α^* von Monomen mit höchstens $\lceil \log_2 \frac{t}{\varepsilon} \rceil$ Literalen gibt, die α bis auf den Fehler ε approximiert.

Wir wenden uns jetzt dem **2. Ziel** (dem Entwurf eines Lernalgorithmus, der eine approximierende Disjunktion kurzer Monome für α findet) zu.
Als ersten Ansatz könnten wir versuchen den Lernalgorithmus für $\lceil \log_2 \frac{t}{\varepsilon} \rceil$-$DNF_n$ anwenden. Die vom Algorithmus angeforderten Beispiele werden jedoch leider **inkonsistent** sein, da wir versuchen α^* zu lernen, aber die Beispiele gemäß α klassifiziert sind.
Gibt es vielleicht nur wenige Formeln mit kurzen Monomen, insbesondere dann wenn t klein ist? Wir könnten dann alle Formeln aufzählen und die erste „gute" Formel wählen. Wir spielen diesen Fall für $t = n$ durch.

Wieviele mögliche α^*-Formeln gibt es? Die Anzahl aller Monome mit höchstens $\log_2(\frac{n}{\varepsilon})$ Literalen ist höchstens

$$\binom{n}{\log_2(\frac{n}{\varepsilon})} \cdot 2^{\log_2(\frac{n}{\varepsilon})} = \binom{n}{\log_2(\frac{n}{\varepsilon})} \cdot \frac{n}{\varepsilon},$$

und somit gibt es mindestens

$$2^{\binom{n}{\log_2(\frac{n}{\varepsilon})} \cdot \frac{n}{\varepsilon}} = 2^{\Omega(n^{\log_2 \frac{n}{\varepsilon}})}$$

α^*-Formeln. Damit scheitert auch dieser Ansatz, da wir super-exponentielle Laufzeit benötigen!

Zum letzten Ansatz: α^* ist insgesamt eine kurze Hypothese. Wie finden wir kurze Hypothesen? Durch einen Occam-Algorithmus!

Unser Lernalgorithmus:

1. Speichere s positive und s negative Beispiele in den Mengen S_+ und S_- ab. s wird in Abhängigkeit von ε, δ, n und t bestimmt.
2. Speichere **alle** Monome mit höchstens k Literalen in der Menge M_1. k wird in Abhängigkeit von ε, δ, n und t bestimmt.

3. Bestimme die Menge all der Monome aus M_1, die einen kleinen Fehler auf den negativen Beispielen haben:

$$M_2 = \{m \mid m \in M_1, m \text{ wird von höchstens } \varepsilon^* \cdot s \text{ negativen Beispielen erfüllt } \},$$

   wobei $\varepsilon^* = \frac{\varepsilon}{2w}$, mit $w = t(2\ln(s) + 5)$.

4. Sei h die leere Hypothese. Wiederhole w mal:
   - Bestimme ein Monom m aus M_2, das die meisten Beispiele aus S_+ erfüllt.
   - Setze $h \equiv h \vee m$.
   - Entferne alle von h erfüllten positiven Beispiele aus der Menge S_+.

Im 3. Schritt erzwingen wir für jedes Monom in M_2 einen Fehler von höchstens $\frac{\varepsilon}{2w}$ auf den negativen Beispielen S_-. Da im 4. Schritt nur w Monome gewählt werden, beträgt der Gesamtfehler auf den negativen Beispielen höchstens $\frac{\varepsilon}{2}$. Es bleibt zu zeigen, daß der Fehler auf den negativen *Eingaben* durch ε beschränkt ist.

Der Occam-Ansatz des Lernalgorithmus befindet sich im 4. Schritt: Dort versuchen wir, α durch eine relativ kurze Disjunktion von kurzen Monomen zu approximieren. Es bleibt zu zeigen, daß Schritt 4 auch tatsächlich eine kurze *Approximation* für α findet. Mit anderen Worten, es ist zu zeigen, daß auch der Fehler auf den positiven Eingaben durch ε beschränkt ist.

Analyse des Lernalgorithmus:
Für den **3. Schritt** müssen wir einerseits zeigen, daß alle kurzen Monome mit zu großem Fehler auf den negativen Eingaben entfernt werden. Andererseits muß der 3. Schritt auch gewährleisten, daß Monome mit kleinem Fehler (auf den negativen Eingaben) nicht entfernt werden.

Das Problem ist, daß wir aus Effizienzgründen nicht alle negativen Eingaben überprüfen können und uns deshalb mit einer Stichprobe der negativen Eingaben, und zwar den negativen Beispielen in S_-, behelfen müssen. Es ist also zu zeigen, daß diese Stichprobe ausreicht, um die richtigen Monome zu entfernen. Das folgende Lemma sichert diese Forderungen:

5.1.3 Lemma Setze $k = \lceil log \frac{4w}{\varepsilon} \rceil$ und $s \geq \frac{12w}{\varepsilon}(\ln(|M_1|) + \ln \frac{4}{\delta})$.

a) Mit Wahrscheinlichkeit mindestens $1 - \frac{\delta}{4}$ gilt für jedes $i \in \{1, \ldots, t\}$:
   - m_i ist 2^{-k}-unwichtig für α oder
   - jedes Teilmonom m von m_i mit genau k Literalen gehört zu M_2.

b) Mit Wahrscheinlichleit mindestens $1 - \frac{\delta}{4}$ gilt für jedes $m \in M_2$, daß

$$prob[m(x) = 1 \mid \alpha(x) = 0] \leq \frac{\varepsilon}{w}$$

   ist.

c) Mit Wahrscheinlichkeit mindestens $1 - \frac{\delta}{4}$ gibt es eine Disjunktion α^* von höchstens t Monomen aus M_2 mit
$$prob[\alpha(x) \neq \alpha^*(x)] \leq \frac{\varepsilon}{4}.$$

Beweis a) Sei m ein Teilmonom von m_i mit genau k Literalen. Falls m_i 2^{-k}-wichtig für α ist besagt Lemma 5.1.2, daß
$$prob[m(x) = 1 \mid \alpha(x) = 0] \leq 2^{-k} \leq \frac{\varepsilon}{4w}$$
gilt. Die erwartete Anzahl von Fehlern auf S_- ist somit höchstens $\frac{\varepsilon}{4w} \cdot |S_-|$. m wird nur dann nicht in die Menge M_2 aufgenommen, wenn ein Fehler von mindestens $\frac{\varepsilon}{2w}$ auf S_- festgestellt wird, d.h. wenn $|\{x \mid x \in S_-, m(x) = 1\}| \geq \frac{\varepsilon}{2w} \cdot |S_-|$ gilt. Die Chernoff-Schranke liefert uns:
$$prob[m \notin M_2] \leq e^{-\frac{\varepsilon \cdot |S_-|}{4w \cdot 3}} = e^{-\frac{\varepsilon \cdot s}{12w}}.$$
Wir fordern
$$\underbrace{|M_1|}_{\text{maximale Anzahl an Teilmonomen}} \cdot \underbrace{e^{-\frac{\varepsilon \cdot s}{12w}}}_{\text{Fehlerwahrscheinlichkeit}} \stackrel{!}{\leq} \frac{\delta}{4}$$
und müssen s entsprechend bestimmen. Logarithmieren führt auf
$$-\frac{\varepsilon \cdot s}{12w} + \ln(|M_1|) \leq \ln(\frac{\delta}{4})$$
und dies ist äquivalent mit
$$s \geq \frac{12w}{\varepsilon}(\ln(|M_1|) + \ln(\frac{4}{\delta})).$$

b) Sei $m \in M_2$ ein Monom mit zu großem Fehler, d.h. $prob[m(x) = 1 \mid \alpha(x) = 0] \geq \frac{\varepsilon}{|w|}$. Die erwartete Anzahl inkonsistenter Beispiele ist mindestens $\frac{\varepsilon \cdot s}{w}$. Wenn $m \in M_2$ ist, dann ist m nur auf $\frac{\varepsilon \cdot s}{2w}$ negativen Beispielen inkonsistent (nach Konstruktion von M_2). Wir erhalten mit der Chernoff-Schranke,
$$prob[m \in M_2] \leq e^{-\frac{\frac{1}{4} \cdot \frac{\varepsilon \cdot s}{w}}{2}} = e^{-\frac{\varepsilon \cdot s}{8w}}.$$
Die Behauptung b) wird somit mit Wahrscheinlichkeit höchstens
$$p := |M_1| \cdot e^{-\frac{\varepsilon \cdot s}{8w}}$$
verletzt. Die Ungleichung für s aus dem Beweis von a) garantiert $p \leq \frac{\delta}{4}$, da $e^{-\frac{\varepsilon \cdot s}{8w}} \leq e^{-\frac{\varepsilon \cdot s}{12w}}$ ist.

c) Entferne jedes 2^{-k}-unwichtige Monom m_i aus α. Der resultierende Fehler auf den positiven Eingaben ist höchstens $t \cdot 2^{-k} \leq w \cdot \frac{\varepsilon}{4w} = \frac{\varepsilon}{4}$. Danach ersetzen wir die übrigen Monom aus α durch (beliebige) Teilmonome mit k Literalen. Der Fehler auf den negativen Eingaben ist nach Lemma 5.1.2 höchstens $t \cdot 2^{-k} \leq w \cdot 2^{-k} \leq \frac{\varepsilon}{4}$. Mit Wahrscheinlichkeit mindestens $1 - \frac{\delta}{4}$ gehören aber (nach Teil a)) alle gewählten Teilmonome zu M_2. □

Im **4. Schritt** approximiert der Lernalgorithmus α durch einen „gierigen" Algorithmus: Dieser gierige Algorithmus wählt stets das Monom aus, das gegenwärtig die meisten positiven Beispiele erfüllt. Wir betrachten das Approximationsproblem in einem allgemeinen Rahmen.

Das partielle Überdeckungsproblem:

- Gegeben seien das Universum S, endliche Mengen $S_1, \ldots, S_x$ (mit $\bigcup_{i=1}^{x} S_i \subseteq S$) sowie der Überdeckungsparameter p, mit $0 < p \leq 1$.
- Gesucht ist eine Indexmenge $I \subseteq \{1, \ldots, x\}$ kleinster Größe, so daß $\bigcup_{i \in I} S_i$ eine p-Überdeckung von S ergibt, d.h. $|\bigcup_{i \in I} S_i| \geq p \cdot |S|$ gilt.

Wir wissen mit dem vorherigen Lemma, daß es eine $(1 - \frac{\varepsilon}{4})$-Überdeckung von $\alpha^{-1}(1)$ gibt. Im nächsten Lemma zeigen wir, daß es dann auch eine $(1 - \frac{\varepsilon}{2})$-Überdeckung von S_+ gibt.

Was ist der Zusammenhang zu unserem Approximationsproblem?
Setze das Universum gleich der Menge aller positiven Beispiele, d.h. $S = S_+$. Für jedes $m \in M_2$ definiere die Menge $S_m = \{x \mid x \in S_+, m(x) = 1\}$. Als Überdeckungsparameter wählen wir $p = 1 - \frac{\varepsilon}{2}$. Der 4. Schritt versucht also, die Beispiele in S_+ partiell zu überdecken.

Das partielle Überdeckungsproblem (in der Sprachenversion: „Gibt es eine p-Überdeckung mit $|I|$ Mengen?") ist NP-vollständig. Glücklicherweise gibt es jedoch einen guten Approximationsalgorithmus A, der eine partielle Überdeckung gierig konstruiert:

1. Setze $I := \emptyset$.
2. Solange noch keine p-Überdeckung von S erreicht ist, wähle eine Menge S_i größter Mächtigkeit,
   setze $I := I \cup \{i\}$
   und $S_j := S_j \setminus S_i$ für jedes j, mit $1 \leq j \leq x$.

Es kann gezeigt werden (siehe M. Kearns, M. Li, *Learning in the presence of malicious errors*, Proceedings of the 20th Annual ACM Symposium on the Theory of Computing, pp. 267-279, 1988), daß A eine p-Überdeckung mit höchstens

$$opt(2 \cdot \ln(|S|) + 5)$$

vielen Mengen findet, wobei *opt* die minimale Anzahl von Teilmengen einer p-Überdeckung von S ist. Beachte, daß die Laufzeit von A polynomiell in x und $|S|$ ist.

Zurück zur Analyse des 4. Schritts: Wir kommen mit $opt(2 \cdot \ln(s) + 5)$ Monomen aus, um genügend viele positive Beispiele zu überdecken. Was ist der Wert von *opt*? Es *sollte* $opt \leq t$ gelten, aber wir müssen wiederum den *Stichprobeneffekt* behandeln.

5.1.4 Lemma a) Eine $(1-\frac{\varepsilon}{4})$-Überdeckung von $\alpha^{-1}(1)$ ist mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{2}$ eine $(1-\frac{\varepsilon}{2})$-Überdeckung von S_+.
b) Im 4. Schritt wird mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{2}$ eine $(1-\frac{\varepsilon}{2})$-Überdeckung von S_+ mit höchstens $w = t(2 \cdot \ln(s) + 5)$ vielen Monomen gefunden.

Beweis **a)** Wir geben nur die Beweisskizze an: Bis auf einen Bruchteil von $\frac{\varepsilon}{4}$ werden alle positiven Eingaben von den Monomen in α^* überdeckt. Wir zeigen, daß die Monome in α^*, bis auf einen Bruchteil von $\frac{\varepsilon}{2}$, alle positiven Beispiele überdecken.

Das Ergebnis folgt aus einer Anwendung der Chernoff-Schranke: Der erwartete Bruchteil nicht überdeckter positiver Beispiele ist halb so groß wie der geforderte Bruchteil nicht überdeckter positiver Beispiele.
b) Wir erhalten $opt \leq t$ als eine Konsequenz von Teil a). Der gierige Approximationsalgorithmus findet also, mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{2}$, eine $(1-\frac{\varepsilon}{2})$-Überdeckung von S_+ mit höchstens $opt(2 \cdot \ln(s) + 5) \leq t(2 \cdot \ln(s) + 5) = w$ Monomen. □

Im Verlauf der Analyse sind wir von einer Überdeckung der positiven Eingaben zu einer Überdeckung der positiven Beispiele übergegagen. Wir wollen jedoch α approximieren, d.h. wir brauchen eine Aussage über den Fehler der Überdeckung bzgl. der positiven Eingaben. Als letzter Schritt des Korrektheitsbeweises bleibt also der Nachweis eines Fehlers von höchstens ε auf den positiven Beispielen:

5.1.5 Lemma **a)** Eine $(1-\frac{\varepsilon}{2})$-Überdeckung von S_+ ist mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{2}$ auch eine $(1-\varepsilon)$-Überdeckung von $\alpha^{-1}(1)$.
b) Mit Wahrscheinlichkeit mindestens $1-\delta$ findet obiger Lernalgorithmus eine Hypothese h mit

$$prob[\alpha(x) \neq h(x)] \leq \varepsilon.$$

Beweis **a)** Das Argument wird wiederum mit der Chernoff-Schranke geführt: Diesmal ist der erwartete Bruchteil nicht überdeckter positiver Eingaben $\frac{\varepsilon}{2}$, während nur ein Bruchteil von höchstens ε gefordert wird. Der formale Beweis sei dem Leser überlassen.
b) In Lemma 5.1.3 Teil b) haben wir gezeigt, daß mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{4}$ (und somit erst recht mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{2}$) für jedes Monom $m \in M_2$ gilt, daß $prob[m(x) = 1 \mid \alpha(x) = 0] \leq \frac{\varepsilon}{w}$ ist. Da wir im 4. Schritt w Monome aus M_2 für die Hypothese wählen, ist der Fehler der Hypothese h auf den negativen Eingaben beschränkt durch ε, d.h.

$$prob[h(x) = 1 \mid \alpha(x) = 0] \leq \varepsilon.$$

Teil a) dieses Lemmas besagt, daß der Fehler auf den positiven Eingaben mit Wahrscheinlichkeit mindestens $1-\frac{\delta}{2}$ durch ε beschränkt ist, d.h.

$$prob[h(x) = 0 \mid \alpha(x) = 1] \leq \varepsilon.$$

Diese *beiden* Fehlerschranken gelten mit Wahrscheinlichkeit mindestens $1-\delta$. □

Wir beenden die Analyse des Lernalgorithmus mit einer Abschätzung der Laufzeit bei einem Zielkonzept mit höchstens polynomiell in n vielen Monomen. (Bei beliebiger Anzahl an Monomen gibt es 2^{2^n} viele DNF_n-Formeln. Jeder Algorithmus benötigt bereits für das Aufschreiben des Ergebnisses exponentielle Zeit: $\log_2(2^{2^n}) = 2^n$.)

5.1.6 Satz *Für $t = poly(n)$ und $\delta \geq 2^{-n}$ fordert der Lernalgorithmus $poly(\frac{1}{\varepsilon}, n)$ Beispiele an und läuft in Zeit $n^{O(\log \frac{n}{\varepsilon})}$. Es werden nur Monome der Länge $O(\log \frac{n}{\varepsilon})$ benutzt.*

Beweis Der Lernalgorithmus fordert s positive und s negative Beispiele an, wobei

$$s \geq \frac{12w}{\varepsilon}(\ln(|M_1|) + \ln(\frac{4}{\delta}))$$

gilt. Es ist $w = t(2 \cdot \ln(s) + 5)$ und $t = poly(n)$. Da es höchstens 2^n Eingaben gibt, ist $s \leq 2^n$. Damit folgt

$$\begin{aligned} w &\leq poly(n)(2 \cdot n + 5) \\ &= O(poly(n)). \end{aligned}$$

Die Maximalzahl der Literale pro Monom in der Hypothese ist k, wobei

$$\begin{aligned} k &= \lceil \log \tfrac{4w}{\varepsilon} \rceil \\ &= O(\log n + \log\log(\tfrac{1}{\varepsilon})) \\ &= O(\log(\tfrac{n}{\varepsilon})). \end{aligned}$$

Somit ist $|M_1| \leq (2n)^k = n^{O(\log(\frac{n}{\varepsilon}))}$. Für die geforderte Beispielsanzahl erhalten wir dann

$$\begin{aligned} s &= \Omega\Big(\tfrac{poly(n)}{\varepsilon} \cdot ((\log(\tfrac{n}{\varepsilon}))^2 + \underbrace{\ln(\frac{4}{\delta})}_{\leq n,\ \text{da}\ \delta \leq \frac{1}{2^n}})\Big) \\ &= poly(n, \tfrac{1}{\varepsilon}). \end{aligned}$$

Die Laufzeit ist offensichtlich beschränkt durch $poly(|M_1|, n)$ und damit von der Form $poly(n, n^{O(\log(\frac{n}{\varepsilon}))}) = n^{O(\log(\frac{n}{\varepsilon}))}$. □

5.2 Weitere Ergebnisse für das Lernen von DNF-Formeln

Der gerade vorgestellte Algorithmus ist bisher ungeschlagen, wenn wir aktives Lernen verbieten. Im aktiven Lernmodell lassen sich bessere Ergebnisse erzielen.

Y. Mansour (*An $n^{O(\log\log n)}$ learning algorithm for DNF under the uniform distribution*, Proc. of the 5th Annual ACM Workshop on Computational Learning Theory, pp. 53 - 61, 1992) zeigt, daß in Zeit $n^{O(\log\log n)}$ eine Formel gefunden werden kann, die, bezüglich der

Gleichverteilung, einen vorgeschriebenen Fehler ε nicht überschreitet. Der Algorithmus von Mansour benutzt Elementfragen.

Mit Elementfragen und Äquivalenzfragen können DNF-Formeln mit k Monomen in Zeit $O(n \cdot 2^{O(k)})$ **exakt** bestimmt werden. Dieses Result wird hergeleitet in: A. Blum und S. Rudich, *Fast learning of k-term DNF formulas with queries*, Proc. of the 24th Annual ACM Symposium on Theory of Computing, 1992. Damit sind DNF-Formeln mit logarithmisch vielen Monomen in polynomieller Zeit erlernbar.

Als nächstes betrachten wir das PAC-Modell, aber verstärkt durch die zusätzliche Möglichkeit Elementfragen stellen zu können. Dieses Modell ist mindestens so mächtig, wie das Modell der Element- und Äquivalenzfragen. Wenn eine DNF-Formel (mit polynomiell in n vielen Monomen) im PAC-Sinne (mit Elementfragen) zu erlernen ist, dann werden Elementfragen **nicht** helfen. Dies ist eine Konsequenz der Arbeit: D. Angluin und M. Kharitonov, *When won't membership queries help?*, Proc. of the 23rd Annual ACM Symp. on Theory of Computing, pp. 444 - 454, 1991.

Klevere Fragen können somit nur für DNF-Formeln mit wenigen Monomen gefunden werden.

6 Aktives Lernen

In den bisherigen Kapiteln haben wir gesehen, daß passives Lernen (der Lerner fordert klassifizierte Beispiele an) nicht sehr erfolgreich ist.

In diesem Kapitel erweitern wir das Lernmodell: der Lerner darf Elementfragen und Äquivalenzfragen stellen.

- Bei einer Elementfrage gibt der Lernalgorithmus eine Eingabe (aus dem Eingabebereich) an und erhält die Klassifizierung der Eingabe als Antwort.
- Bei einer Äquivalenzfrage gibt der Lernalgorithmus seine gegenwärtige Hypothese (aus dem Hypothesenraum) an und erhält als Antwort entweder „die Hypothese ist richtig" oder ein Gegenbeispiel (aus dem Eingabebereich).

Wir „praxisnahe" sind die beiden Typen von Fragen?
Eine Menge von klassifizierten Standard-Beispielen (z.B. charakteristische Schriftproben für das Schrifterkennungsproblem) steht im allgemeinen zur Verfügung und kann als Antwort auf die Äquivalenzfrage genommen werden. So haben wir bereits in den Übungsaufgaben gesehen, daß Algorithmen, die nur Äquivalenzfragen stellen dürfen, nicht mächtiger als PAC-Algorithmen sind. Analog kann gezeigt werden, daß Algorithmen, die Element- und Äquivalenzfragen stellen, nicht mächtiger sind als PAC-Algorithmen, die zusätzlich Elementfragen stellen.

Problematisch sind Elementfragen, da der Lerner nach irgendeinem Element, daß nicht in der Beispielsmenge liegt, fragen kann. Die Beantwortung solcher Fragen kann in einigen Fällen die Intervention eines menschlichen Experten erzwingen und wird somit „teuer".

6.1 Endliche Automaten

Wir wissen bereits, daß es keine PAC-Algorithmen gibt, die endliche Automaten in polynomieller Zeit erlernen (wenn das RSA-Verschlüsselungsschema nicht gebrochen werden kann).
Somit ist es auch nicht möglich, polynomielle Lernalgorithmen zu konstruieren, die endliche Automaten mittels Äquivalenzfragen erlernen. Auch Lernalgorithmen, die nur Elementfragen stellen dürfen, können endliche Automaten nicht in Polynomialzeit erlernen (siehe Aufgabe 9.2).

Wir betrachten deshalb Lernmodelle, die Element- **und** Äquivalenzfragen stellen dürfen.

Sei R eine reguläre Sprache über dem endlichen Alphabet Σ. Ein minimaler endlicher Automat für R besitze q Zustände.

Unser Ziel ist die Konstruktion eines Lernalgorithmus, der einen äquivalenten minimalen endlichen Automaten in Zeit $poly(q)$ erlernt. Wir gehen wie folgt vor:

1. Anfangsphase
   Der Lernalgorithmus stellt die Fragen
   $$\varepsilon \in R? \quad \text{und} \quad a \in R? \text{ für alle a} \in \Sigma.$$
   Der Lernalgorithmus beginnt dann die Konstruktion einer 0/1-Matrix M. Anfänglich besitzt M eine Spalte, die mit dem leeren Wort ε indiziert ist. Weiterhin besitzt M eine Zeile für das leere Wort ε und die Zeile a für jeden Buchstaben $a \in \Sigma$. Wir nennen die Zeile von ε eine Zustandszeile und die Zeilen in Σ δ-Zeilen. Zustandszeilen werden die Zustände des zu lernenden Automaten definieren; δ-Zeilen definieren die Zustandsübergänge.

   [Im folgenden bezeichnen wir mit $M[u, v]$ (für $u, v \in \Sigma^*$) den Eintrag der Matrix M in Zeile u und Spalte v. Für Zeile u und Spalte v wird stets gelten, daß
   $$M[u, v] = \begin{cases} 1 & uv \in R \\ 0 & \text{sonst} \end{cases}$$
   ist.]

2. Definitionsphase
   In der Definitionsphase werden solange Elementfragen gestellt, bis die resultierende Matrix M einen konsistenten endlichen Automaten definiert.
   Die Definitionsphase besteht aus zwei Teilphasen:

   2.1 Die Abschlußphase:
   Diese Phase endet, sobald die Abschlußeigenschaft erreicht ist:
   - Für jede Zustandszeile sind alle δ-Zeilen vorhanden.
     (D.h.: Falls zu dem Wort u eine Zustandszeile existiert, muß es für jedes $a \in \Sigma$ eine δ-Zeile zu ua geben.)

  - Für jede δ-Zeile gibt es eine übereinstimmende Zustandszeile.
    (D.h.: Falls es eine δ-Zeile zu ua gibt, dann muß es auch eine Zustandszeile zu ua geben.)

  Solange die Abschlußeigenschaft nicht erfüllt ist, führt der Algorithmus die folgenden Schritte durch:

  - Wenn es eine Zustandszeile zu einer Eingabe v gibt, so daß va für mindestens ein $a \in \Sigma$ keine δ-Zeile in M besitzt, dann wird die fehlenden δ-Zeile (mit Hilfe von Elementfragen) eingefügt.
  - Wenn eine δ-Zeile nicht als Zustandszeile auftaucht, dann wird sie als Zustandszeile hinzugefügt. (Beim Einfügen einer δ-Zeile als Zustandszeile, müssen möglicherweise neue δ-Zeilen eingefügt werden!)

  Wie bereits oben erwähnt, wird dieses Verfahren solange wiederholt, bis die Abschlußeigenschaft gilt. Danach wird zu Phase 2.2 gewechselt.

2.2 Konsistenzphase

  Der Algorithmus überprüft für je zwei identische Zustandszeilen (zu $u, v \in \Sigma^*$), ob es ein $a \in \Sigma$ gibt, so daß die δ-Zeilen von ua und va verschieden sind. Falls es ein solches a gibt, definiert die Matrix keinen konsisten Automaten: Der den Zustandszeilen von u und v entsprechende Zustand hat zwei Nachfolgezustände unter a! Der Algorithmus versucht deshalb, die Zustandszeilen von u und v durch Einfügung einer neuen Spalte zu trennen.

  Wenn die δ-Zeilen von ua und va unterschiedlich sind, dann gibt es eine Spalte x in M, mit $M[ua, x] \neq M[va, x]$. Genau einer der Eingaben uax und vax gehört zur Sprache, während die andere Eingabe zu verwerfen ist. Der Lernalgorithmus fügt eine neue Matrixspalte für das Wort ax hinzu und die Matrix wird mit Hilfe von Elementfragen vervollständigt. Beachte, daß die obige Inkonsistenz jetzt aufgehoben ist, da die Zustandszeilen von u und v sich in der Spalte ax unterscheiden.

  Der Algorithmus wiederholt dieses Verfahren solange bis Konsistenz erreicht ist, d.h. solange bis identische Zustandszeilen auch stets identische δ-Zeilen besitzen.

Die Teilphasen 2.1 und 2.1 werden solange wiederholt, bis gleichzeitig Abschluß und Konsistenz erreicht ist. Dann kann aus der Matrix ein endlicher Automat konstruiert werden:

- Der Anfangszustand q_ε entspricht der Zustandszeile für das leere Wort.
- Für jede Zustandszeile (mit $u \in \Sigma$ indiziert) wird der Zustand q_u geschaffen, wobei die Zustände übereinstimmender Zustandszeilen identifiziert werden.
- Die δ-Zeilen definieren die Übergangsfunktion des endlichen Automaten: Für Zustand q_u und $a \in \Sigma$ gibt es stets eine mit ua indizierte δ-Zeile. Zu dieser δ-Zeile gibt es eine äquivalente Zustandszeile, die mit v indiziert sei. Wir setzen

$$\delta(q_u, a) = q_v.$$

Die Abschlußeigenschaft sichert die Existenz der Zustandszeile zu ua und die Existenz einer identischen Zustandszeile. Die Konsistenzeigenschaft sichert, daß die Definition des Automaten nicht von der Wahl der Zustandszeilen abhängt (wenn es mehrere identische Zustandszeilen gibt).

- Das Wort u indiziere eine Zustandszeile. Dann ist q_u ein akzeptierender (verwerfender) Zustand genau dann, wenn $M[u, \varepsilon] = 1$ ($M[u, \varepsilon] = 0$) ist.

Man mache sich klar, daß der gefundene Automat konsistent mit der Matrix ist: Für jede Zeile u und jede Spalte v gilt

$$M[u, v] = 1 \Leftrightarrow \text{ der konstruierte Automat akzeptiert } uv.$$

Der Algorithmus wechselt in die 3. Phase.

3. Die Testphase
   Der Algorithmus stellt den durch die Matrix M definierten Automaten als Hypothese auf. Wird die Hypothese akzeptiert, haben wir die reguläre Sprache R erfolgreich erlernt. Ansonsten erhalten wir ein Gegenbeispiel $x = x_1 \ldots x_m \in \Sigma^m$.
   Der Lernalgorithmus berechnet alle Präfixe $x^i = x_1 \ldots x_i$ und fügt sie, falls noch nicht vorhanden, als neue Zustandszeilen zur Matrix M hinzu. M wird sodann durch Elementfragen vervollständigt.
   Der Algorithmus wechselt in die 2. Phase.

Laufzeitanalyse:
Sei L die Länge des längsten Gegenbeispiels. Wie wir oben schon angenommen haben, hat der minimale Automat für R q Zustände. Eine wichtige Beobachtung ist, daß der Lernalgorithmus nur minimale Automaten als Hypothesen aufstellt; denn je 2 Zustände entsprechen verschiedenen Zeilen, die Zustände haben also ein unterschiedliches Akzeptanzverhalten!

Nach jeder widerlegten Hypothese wird in der darauffolgenden Konsistenzphase ein neuer Zustand des Automaten gefunden. Somit stellt der Lernalgorithmus höchstens $q - 1$ Äquivalenzfragen.

Es werden höchstens so viele Elementfragen gestellt, wie es Matrixpositionen in M gibt. Deshalb untersuchen wir die Größe von M:

Wieviele Spalten hat die Matrix M?
Spalten werden nur in der Konsistenzphase hinzugefügt. Bei jedem Einfügen einer Spalte wird aber eine neue Zustandszeile entdeckt: M hat höchstens q Spalten.

Wieviele Zeilen hat die Matrix M?
Es genügt, die Anzahl z der Zustandszeilen zu bestimmen, da die Anzahl der δ-Zeilen durch $|\Sigma| \cdot z$ beschränkt ist.
Höchstens L Zustandszeilen werden innerhalb einer Durchführung der Testphase eingefügt. Da die Testphase stets eine neue Zustandszeile findet, ist sie insgesamt für höchstens $L \cdot q$ Zustandszeilen verantwortlich.

In der Definitionsphase werden neue Zustandszeilen nur in der Abschlußphase hinzugefügt und nur dann, wenn eine δ-Zeile keine übereinstimmende Zustandszeile besitzt. Da in diesem Fall ein neuer Zustand entdeckt wurde, fügt die Definitionsphase somit höchstens q Zustandszeilen ein.
Insgesamt besitzt M also höchstens $q + L \cdot q + q \cdot |\Sigma| + L \cdot q \cdot |\Sigma|$ viele Zeilen.

Die Anzahl der Elementfragen ist durch das Produkt der Zeilenanzahl und der Spaltenanzahl beschränkt. Es werden also höchstens

$$\begin{aligned} & q(q + L \cdot q + q \cdot |\Sigma| + L \cdot q \cdot |\Sigma|) \\ = \; & q^2(1 + L + |\Sigma| + L \cdot |\Sigma|) \\ = \; & q^2(1 + L)(1 + |\Sigma|) \end{aligned}$$

Elementfragen gestellt. Die Laufzeit des Lernalgorithmus ist somit polynomiell in L und q.

6.1.1 Satz *Sei R eine reguläre Sprache, die von einem minimalen endlichen Automaten A mit q Zuständen erkannt wird. Der obige Lernalgorithmus stellt höchstens $q - 1$ Äquivalenzfragen und höchstens $q^2(1 + L)(1 + |\Sigma|)$ Elementfragen, wobei L die Länge des längsten Gegenbeispiels ist. Die Laufzeit des Lernalgorithmus für R ist polynomiell in L und q.*

Wir versuchen jetzt, den obigen Lernalgorithmus zu verbessern; wir möchten mit weniger Elementfragen auskommen.

Wir gehen anfänglich genau wie im obigen Algorithmus vor; insbesondere übernehmen wir die Definitionsphase. (Es wird sich allerdings herausstellen, daß die Konsistenzphase überflüssig ist.) Die Testphase gestalten wir anders.

Sei $x = x_1 \dots x_m$ das erhaltene Gegenbeispiel und δ' die Überführungsfunktion der Hypothese. Beachte, daß wir $\delta'(q_0, z)$ sowohl als Zustand als auch als Wort über Σ auffassen können: Das Wort, das die Zustandszeile von $\delta'(q_0, z)$ indiziert (und damit den vermuteten Automaten vom Anfangszustand zum Zustand $\delta'(q_0, z)$ führt).

Für i, mit $1 \leq i \leq m$, sei

$$y_i = \delta'(q_0, x^i)$$

der dem Zustand entsprechenden String. (y_i ist also der Name des Zustands, den der vermutete Automat nach dem Lesen von $x^i = x_1 \dots x_i$ erreicht.) Wir setzen

$$\alpha_i = y_i \; x_{i+1} \dots x_m.$$

Beachte, daß $\delta'(q_0, \alpha_i) = \delta'(q_0, \alpha_{i+1})$ für jedes i, $1 \leq i \leq m-1$, gilt. Für die Überführungsfunktion δ des zu erlernenden Automaten muß es hingegen ein i geben mit

$$\gamma(\underbrace{\delta(q_0, \alpha_i)}_{\substack{\text{„Lasse Hypothese } i \text{ Schritte laufen"} \\ \text{und dann den gesuchten Automaten}}}) \neq \gamma(\underbrace{\delta(q_0, \alpha_{i+1})}_{\substack{\text{„Lasse Hypothese } i+1 \text{ Schritte laufen"} \\ \text{und dann den gesuchten Automaten}}})$$

(wobei γ einen Zustand als akzeptiertend oder verwerfend klassifiziert).
Warum gibt es ein solches i?
Wenn es kein solches i gibt, dann gilt stets $\gamma(\delta(q_0, \alpha_i)) = \gamma(\delta(q_0, \alpha_{i+1}))$ und somit $\gamma(\delta(q_0, x)) = \gamma(\delta(q_0, \alpha_0)) = \gamma(\delta(q_0, \alpha_m)) = \gamma(\delta(q_0, y_m))$. Der zu erlernende Automat würde also x genau dann akzeptieren, wenn y_m akzeptiert würde und x kann kein Gegenbeispiel sein: y_m ist Index einer Zustandszeile von M, somit hat der Lernalgorithmus $M[y_m, \varepsilon]$ nachgefragt und gelangt zur selben Antwort wie der zu erlernende Automat.

Angenommen, wir haben ein i gefunden mit

$$\gamma(\delta(q_0, \alpha_i)) \neq \gamma(\delta(q_0, \alpha_{i+1})). \qquad (*)$$

Wir wissen, daß einerseits $\delta'(y_i, x_{i+1}) = y_{i+1}$ ist. Andererseits liefert im zu lernenden Automaten y_i (als Zustand) mit Eingabe $x_{i+1}\ x_{i+2} \dots x_m$ ein anderes Ergebnis als y_{i+1} (als Zustand) mit Eingabe $x_{i+2} \dots x_m$. Somit hat die Hypothese die Zustände y_{i+1} und $\delta'(y_i, x_{i+1})$ fälschlicherweise miteinander identifiziert.
Wir korrigieren diesen Irrtum, indem wir das Wort $x_{i+2} \dots x_m$ als **neue Spalte** hinzufügen: Diese neue Spalte erzwingt in der Abschlußphase, daß $\delta'(y_1, x_{i+1})$ zu einer neuen Zustandszeile wird, da diese δ-Zeile keiner Zustandszeile entspricht.

Beachte, daß Q (die Menge der Zustandszeilen) stets aus **verschiedenen** Zeilen besteht: Die Konsistenzphase ist deshalb überflüssig.

Um den neuen Algorithmus abzuschließen, müssen wir zeigen, wie wir ein i mit der Eigenschaft $(*)$ finden können.

3. <u>Die neue Testphase</u>
   Falls wir ein Gegenbeispiel $x = x_1 \dots x_m \in \Sigma^m$ erhalten, wissen wir, daß

   $$\gamma(\delta(y_0, x)) = \gamma(\delta(q_0, \alpha_0)) \neq \gamma(\delta(q_0, \alpha_m))$$

   ist. O.B.d.A gelte $\gamma(\delta(q_0, \alpha_0)) =$ verwerfend und $\gamma(\delta(q_0, \alpha_m)) =$ akzeptierend. Wir suchen ein i mit Eigenschaft $(*)$ durch binäre Suche:

   - Wenn $\gamma(\delta(q_0, \alpha_{\frac{m}{2}})) \cong$ verwerfend, dann stelle eine Elementfrage für das Wort $\alpha_{\frac{3m}{4}}$.
   - Wenn $\gamma(\delta(q_0, \alpha_{\frac{m}{2}})) \cong$ akzeptierend, dann stelle eine Elementfrage für das Wort $\alpha_{\frac{m}{4}}$.

   usw.
   Mit $\log_2 L$ Elementfragen wird dann ein i gefunden, mit

   $$\gamma(\delta(q_0, \alpha_i)) \neq \gamma(\delta(q_0, \alpha_{i+1})).$$

   Füge $x_{i+2} \dots x_m$ als neue Spalte ein. Gehe zur Definitionsphase (hierbei ist keine Konsistenzphase notwendig, da je zwei Zustandszeilen verschieden sind).

Analyse des neuen Algorithmus:
Alle Zustandszeilen sind verschieden und entsprechen damit tatsächlich verschiedenen Zuständen. Somit besitzt die Matrix M höchstens $\underbrace{q}_{\#\text{ Zustandszeilen}} + \underbrace{q \cdot |\Sigma|}_{\#\ \delta\text{-Zeilen}}$ viele Zeilen.

Spalten werden nur in der Testphase eingefügt. Für jedes Gegenbeispiel wird eine neue Spalte eingefügt. Diese neue Spalte führt in der folgenden Abschlußphase zur Entdeckung einer neuen Zustandszeile: Somit wird die Testphase höchstens $q-1$ mal aufgerufen. Damit besitzt M höchstens q Spalten (die Spalte für das leere Wort ε und die übrigen $q-1$ Spalten).

Wieviele Elementfragen werden insgesamt gestellt?
Um die Matrix M vollständig zu definieren, genügen

$$(q + q|\Sigma|) \cdot q = q^2(1 + |\Sigma|)$$

Elementfragen. Jede der höchstens $q-1$ Testphasen benötigt weitere $log_2 L$ Elementfragen.

6.1.2 Satz *Sei R eine reguläre Sprache, die von einem minimalen endlichen Automaten mit q Zuständen erkannt wird. Der zweite Lernalgorithmus erlernt R in Zeit $poly(L, q)$. Es werden höchstens $q-1$ Äquivalenzfragen und höchstens $q^2(1 + |\Sigma|) + (q-1)\log_2 L$ Elementfragen gestellt.*

6.2 Lernen ohne Reset

Im Kapitel 6.1 haben wir implizit angenommen, daß wir stets zum Anfangszustand zurückkehren können: ansonsten können Elementfragen nicht beantwortet werden. Wir behandeln jetzt das schwierigere Problem des Lernens ohne „Reset“.

Zuerst führen wir den Begriff einer Moore-Maschine ein.

6.2.1 Definition **a)** Eine Moore-Maschine wird beschrieben durch die Komponenten $(V, E, aktion, resultat, \Sigma, \Gamma)$. Hierbei gilt:

- (V, E) ist ein gerichteter Graph.
- Σ und Γ sind endliche Mengen.
- Die Funktion *resultat* markiert die Knoten aus V: $resultat : V \to \Gamma$.
- Die Funktion *aktion* markiert die Kanten aus E: $aktion : E \to \Sigma$, wobei es für jeden Knoten $v \in V$ und für jedes $\alpha \in \Sigma$ genau eine Kante $(v, w) \in E$ gibt, die mit α markiert ist.

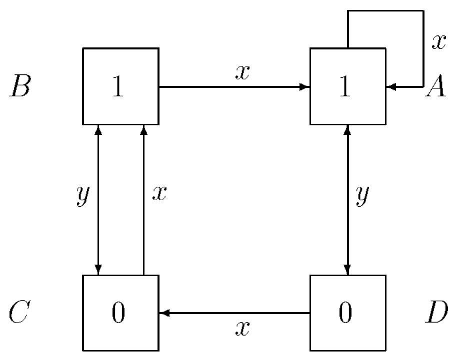
b) Für $a \in \Sigma^*$ bezeichnen wir die Folge der Resultate auf dem durch a beschriebenen Weg mit Anfangsknoten v mit $resultat(v, a)$:

$$resultat(v, a) = (resultat(v), resultat(v_{a_1}), resultat(v_{a_1 a_2}), \ldots, resultat(v_a)),$$

wobei $resultat(v_{a_1 \ldots a_i}) \cong$ Resultat des Endknotens, der von v aus über den durch $a_1 \ldots a_i$ beschriebenen Weg erreicht wird.

Beispiele

1. Deterministische endliche Automaten sind Moore-Maschinen, wobei:
   - Die Knotenmenge der Zustandsmenge entspricht,
   - $\Gamma = \{0, 1\}$ (also nicht-akzeptierend bzw. akzeptierend) ist,
   - Σ dem Eingabealphabet entspricht, und
   - die Kantenmenge E durch die Überführungsfunktion δ spezifiziert wird.
2. Der folgende Graph definiert eine Moore-Maschine:



3. Die Kreuzworträtselwelt:
   Ein Roboter wird auf ein Quadrat eines leeren Kreuzworträtsels gesetzt. In einem Schritt kann der Roboter eine der Aktionen {gehe vorwärts, dreh dich um 90° nach rechts, dreh dich um 90° nach links} ausführen. Das Resultat seiner Aktion ist ein Element aus Γ = {Hindernis, Wand}. Das Resultat ist „Hindernis", wenn er in der Ferne gegen ein schwarzes Quadrat schaut, sonst ist „Wand" das Resultat. Es ist dem Roboter nicht erlaubt, das Kreuzworträtsel zu verlassen: Die Aktion „gehe vorwärts" wird in einem solchen Fall nicht ausgeführt.

Wir versuchen jetzt das folgende Problem zu lösen: Gegeben sei eine unbekannte Moore-Maschine. Ein Roboter wird auf einen Knoten v aus V gesetzt. Sein „Überlebensziel" ist die Entwicklung eines „adäquaten" Modells der Maschine.

Um zu lernen führt der Roboter eine Aktion $a \in \Sigma$ aus: Er wandert von v über die mit a markierte Kante (v, w) zum Knoten w. Als Resultat seiner Aktion erfährt er die Knotenmarkierung $resultat(w)$. Der Roboter setzt seinen Weg fort, um die ganze Maschine zu erkunden. Dabei kann er aber nur in der vorgeschriebenen Richtung laufen; der Schritt zurück (und damit ein Reset) ist unmöglich.

Das Ziel des Roboters ist die Voraussage des Resultats seiner Aktionen. Wenn ein Reset möglich wäre, könnten wir die Resultate aus Kapitel 6.1 anwenden: Der Roboter versucht, einen endlichen Automaten mit Ausgabe zu erlernen.

Wir werden im folgenden zeigen, wie wir sogar ohne Reset erfolgreich lernen können. Wir müssen dabei aber annehmen, daß der Graph stark zusammenhängend ist, d.h. zwischen je zwei Knoten muß ein Pfad vom ersten zum zweiten Knoten existieren.

Zentral für unseren Lernalgorithmus ist der Begriff einer Homing-Sequenz:

6.2.2 Definition Sei $(V, E, aktion, resultat, \Sigma, \Gamma)$ eine Moore-Maschine.
a) Für eine Aktionsfolge $a \in \Sigma^*$ sei $ende(v, a)$ der Endknoten des durch a beschriebenen Weges mit Anfangsknoten v.
b) Ein Wort $h \in \Sigma^*$ heißt genau dann Homing-Sequenz, wenn für alle $v, w \in V$ gilt:

$$resultat(v, h) = resultat(w, h) \Rightarrow \; ende(v, h) = ende(w, h).$$

Eine Homing-Sequenz erlaubt also, den Endknoten eines Weges zu identifizieren, wenn die Folge der Resultate des Weges bekannt ist.

Beispiel Für die Moore-Maschine aus dem 2. Beispiel ist

$$resultat(A, xy) = 110 \;\; \text{und} \;\; \mathrm{ende(A, xy) = D}.$$

Die Einzelaktion x ist bereits eine Homing-Sequenz: Aus der Resultatfolge 00 (bzw. 01, 11) kann geschlossen werden, daß der Knoten C (bzw. B, A) Endknoten ist. Knoten D kann offensichtlich als Endknoten nicht auftreten.

6.2.3 Lemma Sei M eine minimale Moore-Maschine mit q Knoten. Dann besitzt M eine Homing-Sequenz der Länge höchstens $(q-1)^2$.

Beweis Wir finden eine Homing-Sequenz h mit der folgenden Prozedur:

1. Setze $h = \varepsilon$.

2. Solange es Knoten v und w gibt mit

$$resultat(v, h) = resultat(w, h), \text{ aber } ende(v, h) \neq ende(w, h):$$

Suche eine kürzeste Aktionsfolge $a \in \Sigma^*$ mit $resultat(v, ha) \neq resultat(w, ha)$.

Warum existiert eine solche Aktionsfolge a?
Es gilt $ende(v,h) \neq ende(w,h)$, d.h. wir befinden uns auf zwei verschiedenen Knoten k_1 und k_2. Falls für alle Aktionsfolgen $a \in \Sigma^*$ gilt:

$$resultat(\underbrace{ende(v,h)}_{k_1}, a) = resultat(\underbrace{ende(w,h)}_{k_2}, a),$$

dann ist M nicht minimal. Somit gibt es ein a, mit

$$\begin{array}{rrcl} & resultat(ende(v,h), a) & \neq & resultat(ende(w,h), a) \\ \Rightarrow & resultat(v, ha) & \neq & resultat(w, ha). \end{array}$$

Wie lang ist eine solche Aktionsfolge a?
Jedes a mit $|a| > q-1$ durchläuft eine Schleife. Somit gibt es ein $a \in \Sigma^{\leq q-1}$ mit

$$resultat(ende(v,h), a) \neq resultat(ende(w,h), a).$$

Wie oft ist die Schleife im 2. Schritt zu durchlaufen?
In jeder Iteration wächst die Menge

$$\{resultat(v,h) \mid v \in V\}$$

um mindestens ein Element. Da die Menge höchstens $|V| = q$ Elemente enthalten kann, durchlaufen wir höchstens $q-1$ Iterationen.

Somit erhalten wir mit obiger Prozedur eine Homing-Sequenz der Länge höchstens $(q-1)^2$.
□

Sei M die zu lernende Moore-Maschine. Wir nehmen zuerst an, daß der Roboter eine Homing-Sequenz h bereits kennt.
Welche Vorteile bringt eine Homing-Sequenz?
Wenn wir die Homing-Sequenz auf einen beliebigen Knoten $v \in V$ anwenden, erhalten wir eine Resultat-Sequenz σ. Wann immer das Resultat σ ist, wissen wir, daß wir zum selben Endknoten zurückgekehrt sind. Damit haben wir fast einen Reset erreicht. (Wie bereits erwähnt können wir bei Moore-Maschinen mit Reset die Ergebnisse aus Kapitel 6.1 benutzen.)
Leider gibt es aber keine Garantie, daß die Resultat-Sequenz σ häufig genug auftritt!

Wir behelfen uns dadurch, daß wir für jede Resultat-Sequenz σ eine Kopie L_σ des Lernalgorithmus aus Satz 6.1.2 bereitstellen. L_σ interpretiert die zu lernende Moore-Maschine M als einen endlichen Automaten mit dem durch σ spezifizierten Anfangszustand. Wenn wir eine Homing-Sequenz h kennen, ist somit der folgende Algorithmus erfolgreich:

1. Führe h aus und erhalte σ als Resultat. Wenn σ bisher noch nicht erhalten wurde, dann initialisiere den Lernalgorithmus L_σ.

2. Simuliere die nächste Frage von L_σ: Ist dies eine Elementfrage, dann führe die entsprechende Aktionsfolge $a \in \Sigma^*$ durch. Ist die nächste Frage eine Äquivalenzfrage, dann simuliere einen Äquivalenztest.

3. Wenn ein erfolgreicher Äquivalenztest durchgeführt wurde, beende den Algorithmus. Ansonsten gehe zum 1. Schritt.

(Wir können im 2. Schritt des Algorithmus jeweils nur eine Elementfrage simulieren, da der Roboter nach dem Ausführen der entsprechenden Aktionsfolge auf irgendeinem Knoten des Graphen landet. Wenn er von diesem Knoten aus die Homing-Sequenz durchläuft, erhält er eine Resultatfolge und kann dann in der (dieser Resultatfolge) entsprechenden Kopie weiterarbeiten.)

Analyse:
Höchstens $|V| = q$ Kopien des Lernalgorithmus L werden simultan bearbeitet. Jede Kopie erfordert höchstens $q^2(1 + |\Sigma|) + q \log_2 L$ Elementfragen und höchstens $q - 1$ Äquivalenzfragen. Jede „Element“- oder „Äquivalenzfrage“ wird durch eine Anwendung der Homing-Sequenz gefolgt. Damit ist der Gesamtaufwand

$$O\left(|h| \cdot \left(q^3(1 + |\Sigma|) + q^2 \log_2 L\right)\right).$$

Als letztes Problem bleibt die Bestimmung einer Homing-Sequenz:
Wir nehmen an, daß eine Homing-Sequenz h bereits gefunden ist. (Am Anfang setzen wir $h = \varepsilon$.) Mit diesem h lassen wir den obigen Algorithmus laufen. Wenn eine Inkonsistenz gefunden wird, dann wissen wir, daß h keine Homing-Sequenz ist. Wir verlängern h entsprechend der Inkonsistenz und zerstören alle Kopien von L. (Diese sind nutzlos, da sie nicht mit einer Homing-Sequenz erstellt wurden.) Danach wiederholen wir den Algorithmus mit dem verlängerten h (als angenommene Homing-Sequenz).

Von welcher Art kann eine Inkonsistenz sein?
Wir nehmen an, daß wir die Anzahl der Knoten $|V| = n$ kennen. (Wenn nicht, gehe nacheinander die verschiedenen Möglichkeiten $n = 1, n = 2, \ldots$ durch.) Wir wissen, daß h keine Homing-Sequenz ist, wenn eine der Kopien L_σ entdeckt, daß $n + 1$ Zustände vorhanden sind.
In einem solchen Fall ist unsere bisherige Vorarbeit wertlos, und wir zerstören alle Kopien.
Aber wie korrigieren wir h?
Sei L_σ die Kopie, die eine Fehlermeldung aufdeckt. Dann besitzt die Matrix M_σ [5] die Zustandszeilen

$$s_1, \ldots, s_{q+1}.$$

Angenommen, wir erreichen σ nach Anwendung von h auf den (unbekannten) Zustand q der zu lernenden Moore-Maschine M. Da es nur q Zustände in M gibt, muß es i, j (mit $i \neq j$) geben, so daß

$$ende(q, hs_i) = ende(q, hs_j) \qquad (*)$$

[5] M_σ ist die Martix, die der Lernalgorithmus L_σ aufbaut

gilt. Andererseits sind s_i und s_j aber verschiedene Zeilen in M_σ. Somit gibt es mindestens eine Spalte t mit

$$M_\sigma[s_i, t] \neq M_\sigma[s_j, t]. \qquad (**)$$

Der zu lernende Automat wird aber, mit q als Anfangszustand, sowohl unter hs_it als auch unter hs_jt dasselbe **End**ergebnis berechnen:

$$resultat(ende(q, hs_it)) = resultat(ende(q, hs_jt)) =: \gamma$$

(dies ist eine Konsequenz von $(*)$). Somit gilt entweder $M_\sigma[s_i, t] \neq \gamma$ oder $M_\sigma[s_j, t] \neq \gamma$. Angenommen es ist $M_\sigma[s_i, t] \neq \gamma$, dann ist hs_it unsere neue „Homing-Sequenz".
Warum ist diese Sequenz besser als die vorherige?
Die Elementfrage s_it führt vom Zustand qh ausgehend auf die abweichende Antwort γ. Die vorher erhaltene Antwort $M_\sigma[s_i, t]$ wurde also von einem anderen Zustand q^*h, mit $q \neq q^*$, aus erhalten. Die Sequenz hs_it separiert die vorher gleichbehandelten Zustände q und q^*: Es gilt $resultat(ende(q, h)) = \sigma = resultat(ende(q^*, h))$ und $resultat(ende(q, hs_it)) \neq resultat(ende(q^*, hs_jt))$, da sich die Resultat-Sequenzen in der letzten Komponente unterscheiden. Somit haben wir mindestens eine Resultat-Sequenz mehr erhalten. Da es insgesamt höchstens q Resultat-Sequenzen gibt, ist dies ein Fortschritt: Höchstens $q-1$-mal müssen wir neu starten.

Leider kennen wir q nicht, und damit sind uns auch s_i und s_j unbekannt. Die Lösung dieses Problems ist: Wir wählen i und j zufällig, bestimmen ein t mit der Eigenschaft $(**)$ und wählen einen der beiden Zustände s_i und s_j zufällig (z.B. s_i). Dann starten wir den Test s_it. Mit Wahrscheinlichkeit $\frac{1}{2} \cdot \binom{n+1}{2}^{-1} = \frac{1}{n(n+1)}$ sind wir erfolgreich und versuchen unser Glück mit einem neuen, verbesserten Kandidaten h.

Wenn wir nicht erfolgreich sind, wenden wir die alte Sequenz h wieder an. Sobald wir im 1.Schritt des Algorithmus wieder σ erhalten, wiederholen wir das Experiment. Solange wir eine von σ verschiedene Resultatfolge im 1. Schritt erhalten, arbeiten wir in der entsprechenden Kopie von L weiter, als sei nichts geschehen.
Kurz über lang erhalten wir σ wieder oder entdecken in den Kopien für andere Resultatfolgen Inkonsistenzen.

Der endgültige Algorithmus:

1. Setze $h = \varepsilon$ und lese q ein.
2. Wiederhole solange bis eine Äquivalenzfrage erfolgreich ist:
   - 2.a Wende h an und erhalte die Resultatfolge σ. Initialisiere eine Kopie L_σ, falls eine solche Kopie noch nicht existiert.
   - 2.b Wenn L_σ bisher höchstens q Zustände gefunden hat, dann simuliere die nächste Frage von L_σ. Der Lernalgorithmus hält, wenn eine erfolgreiche Äquivalenzfrage gestellt wurde.

2.c Wenn L_σ bereits $q+1$ Zustände gefunden hat, dann
   - wähle zufällig 2 Zeilen s_i und s_j aus M_σ,
   - bestimme eine sie trennende Spalte t in M_σ,
   - wähle zufällig ein $p \in \{s_i, s_j\}$ und lasse den Test pt laufen. Wenn ein neues Ergebnis gefunden wird, dann
     * Zerstöre alle Kopien von L und
     * wähle hpt als neue Homing-Sequenz

Analyse:
Höchstens $(q-1)$-mal werden Kopien von L vernichtet. Zwischen zwei Vernichtungsaktionen werden höchstens

$$O((q^3 + q^2 \log_2 L) \cdot |h|)$$

Elementfragen gestellt, wobei $|h|$ die Länge der gegenwärtigen Homing-Sequenz bezeichnet. Nach $O(q^3 + q^2 \log_2 L)$ Elementfragen wird in jeder Kopie eine Inkonsistenz gefunden. Ihre Beseitigung gelingt mit Wahrscheinlichkeit $\frac{1}{2}$, wenn eine inkonsistente Kopie $q(q+1)$-mal aufgesucht wurde.
Durch $(\log_2 \frac{n^2}{\delta})$-malige Wiederholung erreichen wir mit Wahrscheinlichkeit $1-\delta$, daß die Moore-Maschine korrekt identifiziert wird: Wir haben es mit maximal q^2 Kopien zu tun, die irgendwann eine Inkonsistenz melden. Mit Wahrscheinlichkeit $1-\delta$, haben wir in jeder Kopie Erfolg.

6.2.4 Satz *Sei $0 \le \delta < 1$ gegeben. Die zu lernende Moore-Maschine M wird, mit Wahrscheinlichkeit $1-\delta$, nach höchstens*

$$O\left(\left(q^6 + q^5 \log_2 L\right) |h| \log_2\left(\frac{q}{\delta}\right)\right)$$

„Elementfragen" erlernt, wobei $|h|$ die Länge der endgültigen Homing-Sequenz bezeichnet.

7 Entscheidungsbäume

In den Übugen (Aufgabe 5.2) haben wir polynomielle PAC-Algorithmen für das Erlernen von **Entscheidungslisten** kennengelernt. Wir wenden uns jetzt dem schwierigeren Problem des Lernens von **Entscheidungsbäumen** zu.

Wir besprechen zuerst einen „fast-effizienten" PAC-Algorithmus und werden dann effiziente Algorithmen in einem aktiven Lernmodell kennenlernen.

7.1 Ein „fast-effizienter" PAC-Algorithmus

Wir führen zuerst ein Komplexitätsmaß für binäre Bäume ein, nämlich den Rang binärer Bäume.

7.1.1 Definition Der Rang eines binären Baumes T (Notation: $rang(T)$) ist induktiv definiert:
a) Der Rang eines Baumes, der nur aus der Wurzel besteht, ist 0.
b) Sei T ein binärer Baum mit linkem Teilbaum T_0 und rechtem Teilbaum T_1. Sei $r_i = rang(T_i)$ für $i \in \{1,2\}$. Wenn $r_1 = r_2$, dann ist $rang(T) := r_1 + 1$; ansonsten, wenn $r_1 \neq r_2$, dann ist $rang(T) := \max\{r_1, r_2\}$.

7.1.2 Lemma **a)** Sei k die Anzahl der Knoten eines binären Baums T vom Rang r. T sei reduziert, d.h. keine der n Variablen taucht mehr als einmal auf dem Weg von der Wurzel zu einem Blatt auf. Dann gilt:

$$2^{r+1} - 1 \leq k \leq 2\left(\frac{e \cdot n}{r}\right)^r.$$

b) Sei F_n^r die Menge aller Entscheidungsbäume vom Rang r mit Entscheidungsfunktionen $x_1, \ldots, x_n$. Dann ist für $r \leq n$:

$$|F_n^r| \leq (32n)^{(e \cdot n/r)^r}.$$

c) Ein Entscheidungsbaum mit s Knoten hat höchstens einen Rang von $\lfloor \log_2 s \rfloor$.

Beweis **a)** Der kleinste Entscheidungsbaum vom Rang r ist der vollständige binäre Baum der Tiefe r. Deshalb gilt $2^{r+1} - 1 \leq k$.
Sei $b(n,r)$ die maximale Anzahl von Blättern eines reduzierten Entscheidungsbaumes vom Rang r (mit Entscheidungsfunktionen $x_1, \ldots, x_n$) . Dann gilt offensichtlich:

$$b(0,r) = 1 \text{ und } \mathrm{b(n,0)} = 1.$$

Außerdem ist

$$b(n,r) \leq b(n-1,r) + b(n-1,r-1),$$

denn: Die Anzahl der Blätter eines Baumes wird maximiert, wenn (z.B.) der linke Teilbaum Rang r und der rechte Teilbaum Rang $r-1$ besitzt. Auf diese Weise hat der Baum noch immer Rang r. Der Übergang von n Entscheidungsfunktionen zu $n-1$ Entscheidungsfunktionen ist nötig, da die Funktion der Wurzel nicht mitzuzählen ist.
Eine Lösung der Rekurrenz, mit $\leq$ ersetzt durch $=$, ist durch

$$b(n,r) = \sum_{i=0}^{r} \binom{n}{i}$$

gegeben. Da die Anzahl der Knoten durch die zweifache Anzahl der Blätter beschränkt ist, erhalten wir $k \leq 2\left(\frac{e \cdot n}{r}\right)^r$.
b) Es gibt höchstens 4^m binäre Bäume mit m Knoten: Für jeden Knoten gibt es 4 Wahlmöglichkeiten (kein Kind, nur ein linkes Kind, nur ein rechtes Kind, zwei Kinder).

Es gibt also nur 4^{2i-1} binäre Bäume mit i Blättern. Da jedes der i Blätter klassifiziert wird und jedem inneren Knoten eine der n Entscheidungsfunktionen zugewiesen wird, gibt es somit höchstens $2^i n^{i-1} 4^{2i-1}$ Entscheidungsbäume mit i Blättern.
Mit a) wissen wir, daß ein Baum vom Rang r höchstens

$$m := \left(\frac{en}{r}\right)^r$$

Blätter besitzt. Deshalb ist

$$\begin{aligned} |F_n^r| &\leq \sum_{i=1}^{m} 2^i \cdot n^{i-1} \cdot 4^{2i-1} \\ &\leq \tfrac{1}{4n} \sum_{i=1}^{m} (32n)^i \\ &\leq \tfrac{1}{4n} \cdot \tfrac{(32n)^{m+1}-1}{32n-1} \\ &\leq (32n)^m, \end{aligned}$$

(denn $4n \cdot (32n-1) \geq 32n$).
c) Ein binärer Baum mit Rang $r > \lfloor log_2 s \rfloor$ hat nach Teil a) mindestens $2s-1$ Knoten und damit mehr als s Knoten. □

Eine effiziente Lösung des Konsistenzproblems für Entscheidungsbäume ist bisher nicht bekannt. Wir werden aber sehen, daß das Konsistenzproblem für F_n^r effizient lösbar ist. Deshalb ersetzen wir das Problem des Lernens von Entscheidungsbäumen polynomieller Größe durch das Lernen von Entscheidungsbäumen von logarithmischem Rang (nach Lemma 7.1.2 ist $Decision_{n,poly(n)} \subseteq F_n^{O(\log n)}$). Leider ist dieses Vorgehen sehr teuer, denn:

- Für $r = O(\log n)$ kann gezeigt werden, daß die VC-Dimension von F_n^r mindestens $n^{\Omega(r)}$ beträgt.
- Es gilt $\log_2 |Decision_{n,s}| \leq log_2(2^{\frac{s}{2}} \cdot n^{\frac{s}{2}} \cdot 4^s) = \frac{s}{2} log_2(32n)$. Die VC-Dimension von $Decision_{n,s}$ ist somit durch $O(s \log_2 n)$ beschränkt und deshalb polynomiell, wenn $s = poly(n)$ ist.

<u>Ein Lernalgorithmus für F_n^r:</u>

1. Fordere $\frac{1}{\varepsilon}\left(\log_2 |F_n^r| + \ln(\frac{1}{\delta})\right)$ Beispiele an. Diese Beispiele werden in der Menge S gesammelt. Für $i = 1, \ldots, n$ setze
$$S_0^i = \{x \in S \mid x_i = 0\} \quad \text{und} \quad S_1^i = \{x \in S \mid x_i = 1\}.$$
2. Rufe die Prozedur $Find(S, r)$ auf, um das Konsistenzproblem zu lösen.

Die Prozedur $Find(S, r)$:

1. Wenn alle Beispiele akzeptierend oder alle Beispiele verwerfend sind, dann konstruiere den entsprechenden Baum vom Rang 0 (also eine mit 1 oder 0 klassifizierte Wurzel). Dies beendet $Find(S, r)$.

2. Ansonsten, prüfe ob $r = 0$ ist. Wenn ja, beende $Find(S, r)$ mit der Antwort: „Lösung unmöglich“.

3. Ansonsten ist $r > 0$ und es gibt positive und negative Beispiele. Nenne Variablenposition i informativ, wenn

$$S_0^i \neq \emptyset \quad \text{und} \quad \mathrm{S}_1^{\mathrm{i}} \neq \emptyset$$

gilt. Beginne mit der ersten informativen Position i:

3.a Rufe $Find(S_0^i, r-1)$ und $Find(S_1^i, r-1)$ auf. Wenn beide Aufrufe erfolgreich sind, können wir die beiden erhaltenen Bäume (vom Rang $r-1$) zu einem Baum vom Rang r mit der Entscheidungsfunktion x_i an der Wurzel zusammensetzen. $Find(S, r)$ hält und gibt den gefundenen Baum aus.

3.b Angenommen, ein Aufruf ist gescheitert (z.B. $Find(S_0^i, r-1)$), aber der andere Aufruf ist erfolgreich. In diesem Fall rufe $Find(S_0^i, r)$ auf. Wenn auch dieser Aufruf scheitert, dann beende $Find(S, r)$ mit der Meldung „Lösung unmöglich“. Ansonsten kann $Find(S, r)$ erfolgreich abgeschlossen und der gefundene Baum (mit Teilbäumen vom Rang r und vom Rang $r-1$ und der Entscheidungsfunktion x_i an der Wurzel) ausgegeben werden.

3.c Wenn beide Aufrufe scheitern, dann wechsele zur nächsten informativen Position und gehe zurück zu Schritt 3.a.

4. Der 3. Schritt ist für jede informative Position gescheitert, und die Meldung „Lösung unmöglich“ wird ausgegeben.

Die Prozedur $Find(S, r)$ ist genau der Definition von Bäumen mit Rang r angepaßt: Zuerst wird nach zwei Teilbäumen mit Rang $r-1$ (Schritt 3a) und, wenn notwendig, nach einem Teilbaum mit Rang r und einem Teilbaum mit Rang $r-1$ (Schritt 3b) gesucht. Im Schritt 3b besteht auch die Möglichkeit die Suche einzustellen: Wenn es keinen Teilbaum mit Rang r für (z.B.) S_0^i gibt, dann kann es keinen Teilbaum mit Rang r für S geben. (Denn falls T ein Entscheidungsbaum mit Rang höchstens r für S ist, dann ist T auch ein Entscheidungsbaum mit Rang höchstens r für S_0^i.) In Schritt 3c wird festgestellt, daß x_i als Entscheidungsfunktion ungeeignet ist, und es wird zur nächsten informativen Variable gewechselt.
Find (S, r) ist also korrekt und findet einen Teilbaum mit Rang höchstens r, wenn ein solcher existiert.

Zeitanalyse für $Find(S, r)$:
Sei $T(i, r)$ die maximale Zeit der Find-Prozedur, falls

- eine Beispielsmenge von $\frac{1}{\varepsilon}\left(\log_2 |F_n^r| + \ln\left(\frac{1}{\delta}\right)\right)$ Beispielen vorliegt,
- i Positionen informativ sind,
- und ein Entscheidungsbaum vom Rang r gesucht wird.

Offensichtlich gilt (für $c = O(|S| \cdot n)$) die folgende Rekursion:

$$\begin{array}{rcl} T(0,r) &=& c, \\ T(i,0) &=& c, \\ T(i,r) &=& c + 2iT(i-1,r-1) + T(i-1,r). \end{array}$$

Wenn wir die Rekursion auf den zweiten Summanden $T(i-1,r)$ anwenden, erhalten wir

$$T(i-1,r) \leq c + 2(i-1)T(i-2,r-1) + T(i-2,r).$$

Wir verfahren dann analog mit $T(i-2,r)$, $T(i-3,r)$ und so weiter. Insgesamt erhalten wir also

$$\begin{array}{rcl} T(i,r) &\leq& c \cdot i + 2\sum\limits_{k=1}^{i} kT(k-1,r-1) + c \\ &\leq& c(n+1) + 2\sum\limits_{k=1}^{i} n \cdot T(k-1,r-1) \\ &\leq& c(n+1) + 2nT(i-1,r-1). \end{array}$$

(Wir haben die rekursive Struktur der *Find*-Prozedur und $T(j,r-1) \leq T(j+1,r-1)$ ausgenutzt.) Andererseits ist $T(i-1,r-1) \geq T(0,r-1) = c$ und damit $c(n+1) \leq 2nT(i-1,r-1)$. Es folgt

$$\begin{array}{rcl} T(i,r) &\leq& 4nT(i-1,r-1) \\ &\leq& 4n \cdot 4n \cdot T(i-2,r-2) \\ &\leq& \ldots \end{array}$$

und somit

$$T(i,r) \leq 4^{r-1}n^{r-1}c.$$

7.1.3 Satz **a)** *Der obige Lernalgorithmus für F_n^r ist ein PAC-Algorithmus. Es werden höchstens*

$$\frac{1}{\varepsilon}\left(\left(\frac{e \cdot n}{r}\right)^r \cdot log(32n) + ln\left(\frac{1}{\delta}\right)\right)$$

Beispiele angefordert, und die Laufzeit ist polynomiell in der Größe der Beispielmenge.
b) *Der obige Lernalgorithmus erlernt $Decision_{n,poly(n)}$ mit höchstens*

$$\frac{1}{\varepsilon}\left(n^{O(\log_2 n)} + \ln\left(\frac{1}{\delta}\right)\right) = m$$

Beispielen. Seine Laufzeit ist polynomiell in m.

7.2 Lernen mit Elementfragen

Wir betrachten jetzt Lernalgorithmen, die nur Elementfragen stellen dürfen. Äquivalenzfragen werden nicht gestellt, da bisher nicht bekannt ist, wie wir ein Gegenbeispiel beim Erlernen von Entscheidungsbäumen nutzbringend verwerten können. Der zu entwerfende Lernalgorithmus wird in polynomieller Zeit laufen, aber den gesuchten Entscheidungsbaum nur approximativ bestimmen.
(Tatsächlich werden Entscheidungsbäume gelernt, die gemäß einem inneren Produkt mod 2 verzweigen können.)
Das für uns zentrale Konzept ist die Fourier-Transformation, die wir in den nächsten Schritten einführen.

7.2.1 Definition **a)** Die Funktionen $f, g : \{0,1\}^n \to \mathbb{R}$ seien gegeben. Wir definieren das <u>innere Produkt</u> von f und g (Notation: $< f, g >$) durch

$$< f, g > := \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x) \cdot g(x).$$

Die <u>Norm</u> einer Funktion f (Notation: $||f||$) definieren wir durch

$$||f|| := \sqrt{< f, f >}.$$

b) Wir setzen den Funktionenraum $F_n = \{ f \mid f : \{0,1\}^n \to \mathbb{R} \}$.

F_n ist unter komponentenweiser Addition (Funktionen werden addiert durch (reellwertige) Addition ihrer Funktionswerte) und Skalarmultiplikation ein Vektorraum. Der Vektorraum F_n hat die Dimension 2^n. Offensichtlich ist $<,>$ tatsächlich ein inneres Produkt über F_n. Wir betrachten als nächstes eine spezielle Basis von F_n.

7.2.2 Definition Für $z \in \{0,1\}^n$ sei $\mathcal{X}_z : \{0,1\}^n \to \mathbb{R}$ definiert durch

$$\mathcal{X}_z(x_1, \ldots, x_n) = \begin{cases} +1 & \sum z_i x_i \equiv 0 \bmod 2 \\ -1 & \text{sonst.} \end{cases}$$

Diese Basis ist sogar eine Orthonormalbasis:

7.2.3 Lemma **a)** Die Basis $B = \{\mathcal{X}_z \mid z \in \{0,1\}^n\}$ ist eine Orthogonalbasis, d.h. es gilt $< \mathcal{X}_{z_1}, \mathcal{X}_{z_2} > = \begin{cases} 1 & z_1 = z_2 \\ 0 & \text{sonst.} \end{cases}$

b) Für jedes $f \in F_n$ gibt es eine eindeutige Darstellung $f = \sum_{x \in \{0,1\}^n} a_x \mathcal{X}_x$.

c) Die Koeffizienten a_x heißen Fourierkoeffizienten und es gilt $a_x = < f, \mathcal{X}_x >$.

d) Es gilt die Identität von Parseval, nämlich $||f||^2 = \sum_{x \in \{0,1\}^n} < f, \mathcal{X}_x >^2$.

Beweis **a)** Nach Definition gilt:

$$\begin{aligned} <\mathcal{X}_{z_1}, \mathcal{X}_{z_2}> &= \tfrac{1}{2^n} \sum_{x\in\{0,1\}^n} \mathcal{X}_{z_1}(x)\cdot\mathcal{X}_{z_2}(x) \\ &= \tfrac{1}{2^n} \sum_{<x,z_1>\equiv<x,z_2>\bmod 2} 1 - \tfrac{1}{2^n} \sum_{<x,z_1>\not\equiv<x,z_2>\bmod 2} 1. \end{aligned}$$

Wenn $z_1 = z_2$ ist, dann gilt natürlich stets $< x, z_1 >\equiv< x, z_2 > \bmod 2$ und somit ist $< \mathcal{X}_{z_1}, \mathcal{X}_{z_2} >= 1$. Wir nehmen also $z_1 \neq z_2$ an. Dann ist

$$\{x \mid < x, z_1 >\equiv< x, z_2 > \bmod 2\} = \{x \mid < x, z_1 + z_2 >\equiv 0 \bmod 2\}.$$

Die letzte Menge ist aber ein Vektorraum der Dimension $n-1$ über $\mathbb{Z}/(2)$ und deshalb ist seine Mächtigkeit 2^{n-1}. Mit anderen Worten, die Hälfte aller $x \in \{0,1\}^n$ erfüllt $< x, z_1 >\equiv< x, z_2 >$, die andere Hälfte erfüllt $< x, z_1 >\not\equiv< x, z_2 >$. Als Konsequenz erhalten wir $< \mathcal{X}_{z_1}, \mathcal{X}_{z_2} >= 0$.

b) Die Funktionen in $B = \{\mathcal{X}_z \mid z \in \{0,1\}^n\}$ sind linear unabhängig: Angenommen $\sum_{x\in\{0,1\}^n} \alpha_x \mathcal{X}_x = 0$. Dann ist für jedes y

$$< \mathcal{X}_y, \sum_{x\in\{0,1\}^n} \alpha_x \mathcal{X}_x >= \sum_{x\in\{0,1\}^n} \alpha_x < \mathcal{X}_y, \mathcal{X}_x > \underset{a)}{=} \alpha_y = 0.$$

Da wir 2^n Funktionen in B haben, bilden die Funktionen eine (orthonormale) Basis. Somit gibt es für jedes $f \in F_n$ eine eindeutige Darstellung $f = \sum_{x\in\{0,1\}^n} a_x \mathcal{X}_x$.

c) Wenn $f = \sum_{x\in\{0,1\}^n} a_x \mathcal{X}_x$ gilt, dann ist

$$< f, \mathcal{X}_y >=< \sum_{x\in\{0,1\}^n} a_x \mathcal{X}_x, \mathcal{X}_y >= \sum_{x\in\{0,1\}^n} a_x < \mathcal{X}_x, \mathcal{X}_y >= a_y.$$

d) Es ist $f = \sum_{x\in\{0,1\}^n} a_x \mathcal{X}_x$ mit $a_x =< f, \mathcal{X}_x >$ und deshalb

$$\begin{aligned} ||f||^2 &= < f, f > \\ &= < \sum_{x\in\{0,1\}^n} a_x \mathcal{X}_x, \sum_{y\in\{0,1\}^n} a_y \mathcal{X}_y > \\ &= \sum_{x\in\{0,1\}^n} \sum_{y\in\{0,1\}^n} a_x a_y < \mathcal{X}_x, \mathcal{X}_y > \\ &\underset{a)}{=} \sum_{x\in\{0,1\}^n} a_x^2 \\ &\underset{c)}{=} \sum_{x\in\{0,1\}^n} < f, \mathcal{X}_x >^2 . \end{aligned}$$

$\square$

Warum betrachten wir die Fouriertransformation für Entscheidungsbäume?
Setze $||f||_1 = \sum_x |< f, \mathcal{X}_x >|$.

7.2.4 Lemma Sei T ein Entscheidungsbaum mit b Blättern, der die Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ berechnet. Dann ist

$$||f||_1 \leq b.$$

(Entscheidungsbäume „besitzen" eine kleine L_1-Norm.)

Beweis Sei T ein Entscheidungsbaum der f berechnet. Wir erlauben, daß ein innerer Knoten v mit einem Vektor $E_v \in \{0,1\}^n$ markiert ist. Falls die Eingabe x den Knoten v erreicht und $< x, E_v > \equiv 0 \bmod 2$ (bzw. $< x, E_v > \equiv 1 \bmod 2$) gilt, verzweigen wir nach links (bzw. rechts).
Für einen Knoten v von T sei $t(v)$ die Tiefe von v (also die Länge des längsten Pfades von der Wurzel zu v). Die Menge aller Eingaben, die einen Konten v erreicht bezeichnen wir mit $reach(v) = \{x \in \{0,1\}^n \mid \text{Eingabe } x \text{ erreicht } v\}$. Wir sagen, daß ein Blatt w ein 1-Blatt ist, wenn es ein akzeptierendes Blatt ist.
Wieviele Eingaben erreichen den Knoten v?
Jede Eingabe, die v erreicht, muß $t(v)$ Tests bestanden haben. Wenn wir annehmen, daß der Entscheidungsbaum effizient ist, dann ist **kein** Test eine Linearkombination der anderen Tests. Wenn Eingabe x den Knoten v erreicht, gilt somit

$$< x, E_{v_i} > \equiv b_i \bmod 2 \text{ für alle } i \text{ mit } 1 \leq i \leq t(v),$$

wobei $b_i \in \{0,1\}$ und $E_{v_i} \in \{0,1\}^n$ die Linearkombination des i-ten Vorgängers von v ist. Die Matrix dieses linearen Gleichungssystems hat Rang $t(v)$. Die Menge

$$reach(v) = \{x \mid < x, E_{v_i} > \equiv b_i \bmod 2 \text{ für } 1 \leq i \leq t(v)\}$$

ist also ein affiner Raum der „Dimension" $n - t(v)$. Deshalb erreichen genau $2^{n-t(v)}$ Eingaben einen Knoten der Tiefe $t(v)$.

Wir betrachten als nächstes einen beliebigen Fourierkoeffizienten $< f, \mathcal{X}_z >$.
Was ist der Beitrag der einzelnen Blätter zu $< f, \mathcal{X}_z >$?
Es ist

$$\begin{aligned}
< f, \mathcal{X}_z > &= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)\mathcal{X}_z(x) \\
&= \frac{1}{2^n} \sum_{w \text{ ist 1-Blatt}} \sum_{x \in reach(w)} \mathcal{X}_z(w) \\
&= \sum_{w \text{ ist 1-Blatt}} \frac{1}{2^{t(w)}} \left(\frac{1}{2^{n-t(w)}} \sum_{x \in reach(w)} \mathcal{X}_z(x) \right).
\end{aligned}$$

Für uns ist der Ausdruck

$$beitrag_w(z) := \left| \frac{1}{2^{n-t(w)}} \sum_{x \in reach(w)} \mathcal{X}_z(x) \right|$$

von Interesse.

Behauptung Für jedes Blatt w gibt es genau $2^{t(w)}$ Werte z aus $\{0,1\}^n$ mit

$$beitrag_w(z) = 1.$$

Für die restlichen $2^n - 2^{t(w)}$ Werte von z ist $beitrag_w(z) = 0$.
(Mit anderen Worten, jedes Blatt w trägt zu $2^{t(w)}$ Fourierkoeffizienten mit Absolutbetrag $\frac{1}{2^{t(w)}}$ bei. Der Gesamtbeitrag eines 1-Blattes zur Summe aller Fourierkoeffizienten $||f||_1 = \sum_{z \in \{0,1\}^n} |< f, \mathcal{X}_z >|$ ist somit $2^{t(w)} \cdot \frac{1}{2^{t(w)}} = 1$. (0-Blätter tragen nichts hinzu.) Damit gilt $\sum_{z \in \{0,1\}^n} |< f, \mathcal{Z}_z >| \leq b$, und das Lemma ist somit eine Konsequenz der Behauptung.)

Beweis Sei w ein beliebiges Blatt und seien $v_1, \ldots, v_{t(w)}$ die Knoten auf dem Weg von der Wurzel zum Blatt w. Wenn $x \in reach(w)$, ist somit stets

$$< x, E_{v_i} > \equiv b_i \bmod 2, \qquad (*)$$

für alle i mit $1 \leq i \leq t(w)$.

<u>Fall 1:</u> z ist eine Linearkombination der Vektoren $E_{v_1}, E_{v_2}, \ldots, E_{v_{t(w)}}$.
Offensichtlich gibt es genau $2^{t(w)}$ solche Vektoren z. Wir behaupten, daß stets

$$beitrag_w(z) = \left| \frac{1}{2^{n-t(w)}} \sum_{x \in reach(w)} \mathcal{X}_z(x) \right| = 1$$

gilt. Es genügt zu zeigen, daß alle Elemente aus $reach(w)$ ein konstantes inneres Produkt mit z besitzen: Denn dann ist $\mathcal{X}_z(x)$ entweder 1 oder -1 für alle $2^{n-t(w)}$ Elemente aus $reach(w)$. Da z linear abhängig ist, gilt $z = \sum_{i=1}^{t(w)} \alpha_i E_{v_i}$. Deshalb ist für jedes $x \in reach(w)$, gemäß $(*)$,

$$\begin{aligned} < x, z > &\equiv < x, \sum_{i=1}^{t(w)} \alpha_i E_{v_i} > \\ &\equiv \sum_{i=1}^{t(w)} \alpha_i < x, E_{v_i} > \\ &\underset{(*)}{\equiv} \sum_{i=1}^{t(w)} \alpha_i b_i \bmod 2. \end{aligned}$$

Das innere Produkt ist also tatsächlich konstant.

<u>Fall 2:</u> z ist <u>keine</u> Linearkombination der Vektoren $E_{v_1}, \ldots, E_{v_{t(w)}}$.
Dann ist

$$|\{x \mid x \in reach(w), < x, z > \equiv 1 \bmod 2\}| = |\{x \mid x \in reach(w), < x, z > \equiv 0 \bmod 2\}|.$$

Warum? Die Gleichung $< x, z > \equiv 0$ (wie auch $< x, z > \equiv 1$) ist linear unabhängig von den anderen Gleichungen aus $(*)$.
Damit muß aber gelten, daß $beitrag_z(w) = 0$ gilt. ■

Warum hilft es, daß $||f||_1$ klein ist?
Dann gibt es eine relativ kleine Anzahl von großen Fourierkoeffizienten deren Summe die Funktion f gut approximiert! Formal:

7.2.5 Lemma Sei $f : \{0,1\}^n \to \{-1,1\}$, sei $\varepsilon > 0$ und $groß_\varepsilon = \{z \mid | < f, \mathcal{X}_z > | \geq \frac{\varepsilon}{||f||_1}\}$. Setze

$$g(x) = \sum_{z \in groß_\varepsilon} < f, \mathcal{X}_z > \mathcal{X}_z(x).$$

Dann gilt stets, daß

$$|\{x \mid f(x) \neq \text{ signum(g(x))}\}| \leq \varepsilon \cdot 2^{\mathrm{n}},$$

wobei $\text{signum(y)} = \begin{cases} 1 & y > 0 \\ -1 & \text{sonst} \end{cases}$ ist.

Beweis Zuerst beachte, daß stets

$$\frac{1}{2^n}|\{x \mid f(x) \neq \text{ signum(g(x))}\}| \leq ||\mathrm{f} - \mathrm{g}||^2$$

gilt. Denn $(f(x) - g(x))^2 \geq 1$, falls $f(x) \neq$ signum(g(x)). Es genügt also zu zeigen, daß $||f - g||^2 \leq \varepsilon$ gilt. Aber

$$(f - g)(x) = \sum_{z \notin groß_\varepsilon} < f, \mathcal{X}_z > \mathcal{X}_z(x).$$

Mit der Identität von Parseval (Lemma 7.2.3 d)) ist

$$||f - g||^2 = \sum_{z \notin groß_\varepsilon} < f, \mathcal{X}_z >^2$$

und deshalb

$$\begin{aligned} ||f - g||^2 &\leq \max_{z \notin groß_\varepsilon} | < f, \mathcal{X}_z > | \cdot \textstyle\sum_{z \in \{0,1\}^n} | < f, \mathcal{X}_z > | \\ &\leq \tfrac{\varepsilon}{||f||_1} \cdot ||f||_1 \\ &\leq \varepsilon. \end{aligned}$$

□

Jetzt können wir auch die Strategie des Lernalgorithmus beschreiben: Wenn die Funktion f durch einen Entscheidungsbaum mit wenigen Knoten berechnet wird, dann ist $||f||_1$ relativ klein. Deshalb kann f durch eine Funktion g approximiert werden, die wenige (höchstens $\frac{||f||_1^2}{\varepsilon}$) von Null verschiedene Fourierkoeffizienten besitzt. Zusätzlich ist jeder Fourierkoeffizient relativ groß (mindestens $\frac{\varepsilon}{||f||_1}$) und sollte deshalb „leicht auffindbar" sein. Wie können wir also alle „großen" Fourierkoeffizienten bestimmen? Rekursiv!

7.2.6 Definition Sei $\alpha \in \{0,1\}^k$. Für $f : \{0,1\}^n \to \mathbb{R}$ definieren wir $f_\alpha : \{0,1\}^{n-k} \to \mathbb{R}$ durch

$$f_\alpha(y) := \sum_{z \in \{0,1\}^{n-k}} < f, \mathcal{X}_{\alpha z} > \mathcal{X}_z(y).$$

Die Funktionen f_α „teilen die Fourierkoeffizienten von f unter sich auf": Die Fourierkoeffizienten von f_α stimmen mit den Fourierkoeffizienten von f überein, die α als Präfix haben. Die Suche nach den Fourierkoeffzienten von f ist damit zurückgeführt auf die Suche nach den Fourierkoeffizienten von f_0 und f_1.
Aber hier ist schon ein erstes Problem: Wir können Elementfragen für f stellen, aber nicht für f_0 oder f_1!

Wir zeigen, wie der Wert von $f_\alpha(y)$ durch Elementfragen (bzgl. f) zumindest approximativ bestimmt werden kann:

7.2.7 Lemma Für jedes α mit $\alpha \in \{0,1\}^k$ gilt

$$f_\alpha(x) = \frac{1}{2^k} \sum_{y \in \{0,1\}^k} f(yx)\mathcal{X}_\alpha(y).$$

Beweis Es ist

$$\frac{1}{2^k} \sum_{y \in \{0,1\}^k} f(yx)\mathcal{X}_\alpha(y) = \frac{1}{2^k} \sum_{y \in \{0,1\}^k} \left(\sum_{z_1, z_2} < f, \mathcal{X}_{z_1 z_2} > \mathcal{X}_{z_1 z_2}(yx) \right) \mathcal{X}_\alpha(y),$$

denn wir haben $f(yx)$ durch seine Fouriertransformation ersetzt (wobei $z_1 \in \{0,1\}^k, z_2 \in \{0,1\}^{n-k}$). Aber

$$\begin{array}{rrcl} & \mathcal{X}_{z_1 z_2}(yx) & = & 1 \\ \Leftrightarrow & < z_1 z_2, yx > & \equiv & 0 \bmod 2 \\ \Leftrightarrow & < z_1, y > + < z_2, x > & \equiv & 0 \bmod 2 \\ \Leftrightarrow & \mathcal{X}_{z_1}(y) \cdot \mathcal{X}_{z_2}(y) & = & 1. \end{array}$$

Wir können die rechte Seite der obigen Gleichung also weiter umformen und erhalten:

$$\frac{1}{2^k} \sum_{z_1, z_2} < f, \mathcal{X}_{z_1 z_2} > \mathcal{X}_{z_2}(x) \sum_{y \in \{0,1\}^k} \mathcal{X}_{z_1}(y)\mathcal{X}_\alpha(y).$$

Betrachten wir die innere Summe. Wenn $\alpha \neq z_1$, erhalten wir den Wert 0 als Konsequenz der Orthogonalität und für $\alpha = z_1$ den Wert 2^k. Wir erhalten somit, daß nur Terme mit $\alpha = z_1$ zur Summe beitragen und das Endergebnis ist

$$\sum_{z_2} < f, \mathcal{X}_{\alpha z_2} > \mathcal{X}_{z_2}(x) = f_\alpha(x).$$

□

Wir haben also f_α als einen Erwartungswert beschrieben und approximieren $f_\alpha(x)$ durch die Berechnung von $\frac{1}{m}\sum_{j=1}^{m} f(y_j x)\mathcal{X}_\alpha(y_j)$.
Warum erhalten wir eine gute Approximation?
Mit Chernoff's Schranke sind große Verstöße gegen den Erwartungswert (bei Zufallsvariablen mit Wertebereich $\{-1, 0, 1\}$) höchst unwahrscheinlich! (Wir überlassen eine formale Verifikation dem Leser.)

Wir können die f_α also approximieren und wissen, daß sie die Fourierkoeffizienten von f unter sich aufteilen. Aber wie finden wir die f_α, die die großen Fourierkoeffizienten besitzen?

7.2.8 Lemma Sei Θ eine positive reelle Zahl und sei $f : \{0,1\}^n \rightarrow \{-1,1\}$. Dann gilt: **a)** Für höchstens $\frac{1}{\Theta^2}$ Werte von z ist $| < f, \mathcal{X}_z > | \geq \Theta$.
b) Für jedes k mit $1 \leq k < n$ gibt es höchstens $\frac{1}{\Theta^2}$ Funktionen f_α (mit $\alpha \in \{0,1\}^k$), so daß $\sum_{y \in \{0,1\}^{n-k}} < f, \mathcal{X}_{\alpha y} >^2 \geq \Theta^2$ gilt.

Beweis a) Mit Parseval's Identität folgt, daß $< f, f >= \sum_{z \in \{0,1\}^n} < f, \mathcal{X}_z >^2$ gilt. Andererseits ist $< f, f >= \frac{1}{2^n}\sum_{z \in \{0,1\}^n} f^2(z) \leq 1$ und die Behauptung folgt.
b) Es ist

$$\sum_{\alpha \in \{0,1\}^k} \left(\sum_{y \in \{0,1\}^{n-k}} < f, \mathcal{X}_{\alpha y} >^2 \right) =< f, f >\leq 1$$

und die Behauptung folgt. □

<u>Der Lernalgorithmus:</u>

1. Setze $\Theta = \frac{\varepsilon}{b}$.
   /* Der Lernalgorithmus sucht alle Fourierkoeffizienten mit Absolutbetrag mindestens Θ. Wir wissen, daß ein Entscheidungsbaum mit b Blättern eine L_1-Norm von höchstens b hat. Mit Lemma 7.2.4 approximiert die Summe aller Fourierkoeffizienten (mit Absoultbetrag mindestens $\frac{\varepsilon}{b}$) den Entscheidungsbaum mit Fehler höchstens ε. */

2. Rufe die rekursive Prozedur $Fourier(\varepsilon)$ auf:

   procedure $Fourier(\alpha)$

   - Approximiere $\sum_{y \in \{0,1\}^{n-k}} < f, \mathcal{X}_{\alpha y} >^2$ und erhalte $a(\alpha)$.
   - Wenn $a(\alpha) \leq \frac{\Theta^2}{2}$, dann halte.
   - Ansonsten:
     - Wenn $|\alpha| = n$, dann gib eine Approximation von $< f, \mathcal{X}_\alpha >$ aus und halte. /* Wir haben einen großen Fourierkoeffizienten gefunden. */
     - Wenn $|\alpha| < n$, dann rufe $Fourier(\alpha 0)$ und $Fourier(\alpha 1)$ auf und halte. /* Es ist nicht ausgeschlossen, daß f_α einen großen Fourierkoeffizienten besitzt. Wenn dies der Fall ist, finden wir ihn in $f_{\alpha 0}$ oder $f_{\alpha 1}$. * /

3. Seien $< f, \mathcal{X}_{\alpha_1} >, \ldots, < f, \mathcal{X}_{\alpha_1} >$ die gefundenen Fourierkoeffizienten. Gib

$$h(x) = \sum_{i=1}^{t} < f, \mathcal{X}_{\alpha_i} > \mathcal{X}_{\alpha_i}(x)$$

   als Hypothese aus.

Laufzeitanalyse:
Wir zählen nur die Anzahl der Anrufe von Fourier. Sei die Länge von α fixiert. Nach Lemma 7.2.8 brechen höchstens $\frac{4}{\Theta^2}$ Aufrufe von $Fourier(\alpha)$ nicht ergebnislos ab (wir haben hier vorausgesetzt, daß $a(\alpha)$ bis auf einen additiven Fehler von $\frac{\Theta^2}{4}$ approximiert). Insgesamt, über alle möglichen Längen für α, gibt es somit höchstens $O(n \cdot \Theta^2)$ Aufrufe von $Fourier$.

Korrektheitsbeweis:
Wenn f_α einen großen Fourierkoeffizienten besitzt (mit $| < f, \mathcal{X}_{\alpha z} > | \geq \Theta$), dann ist

$$\sum_{y \in \{0,1\}^{n-k}} < f, \mathcal{X}_{\alpha y} >^2 \geq \Theta^2.$$

Weiterhin gilt für jeden Präfix β von α, daß

$$\sum_{\sigma} < f, \mathcal{X}_{\beta,\sigma} >^2 \geq \sum_{y} < f, \mathcal{X}_{\alpha y} >^2$$

ist. Das heißt, $Fourier(\alpha)$ **wird** irgendwann aufgerufen. Das einzige zu klärende Problem ist die Berechnung der Approximation $a(\alpha)$. Die Identität von Parseval liefert

$$E\left(f_\alpha^2\right) = \sum_{z \in \{0,1\}^{n-k}} < f, \mathcal{X}_{\alpha z} >^2 .$$

Wir haben also den Erwartungswert $E(f_\alpha^2)$ zu approximieren.

f_α ist eine Funktion mit nicht zu großen Werten, denn nach Lemma 7.2.7 ist

$$f_\alpha(x) = E(f(,x) \cdot \mathcal{X}_\alpha)$$

und deshalb

$$|f_\alpha(x)| \leq 1.$$

(Die Funktionen f und $\mathcal{X}_\alpha$ haben Wertebereich $\{-1, 0, 1\}$.) Deshalb erreichen wir eine verläßliche Approximation des Erwartungswertes $E\left(f_\alpha^2\right)$ durch eine Stichprobe $x_1, \ldots, x_s$ geeigneter Größe. Das Resultat ist dann

$$\frac{1}{s} \sum_{i=1}^{s} f_\alpha^2(x_i).$$

Wie aber erhalten wir den Wert von $f_\alpha^2(x_i)$?
Wiederum mit einer Stichprobe geeigneter Größe mit Hilfe von Lemma 7.2.7.

An dieser Stelle tauchen zum ersten und einzigen Mal Elementfragen auf. Mit anderen Worten: Der Lernalgorithmus benutzt Elementfragen nur zur Approximation der Erwartungswerte $E\left(f_\alpha^2\right)$. Äquivalenzfragen werden nicht gestellt!

Es kann durch eine kanonische Anwendung der Chernoff-Schranken gezeigt werden, daß es genügt, f_α an $m_1 = O\left(\frac{1}{\Theta^2} \ln(\frac{n}{\Theta^2 \Theta})\right)$ Stellen auszuwerten, wobei jede Auswertung

$$O\left(\frac{1}{\Theta^2} \ln \frac{m_1}{\Theta^2} \frac{n}{\Theta}\right)$$

Elementfragen für f nach sich zieht.

Damit haben wir insgesamt erhalten:

7.2.9 Satz *Der obige Lernalgorithmus lernt Entscheidungsbäume mit b Blättern, mit Wahrscheinlichkeit mindestens $1 - \delta$ und Fehler höchstens ε (gemäß uniformer Verteilung). Der Lernalgorithmus stellt nur Elementfragen und läuft in Zeit poly $(b, \frac{1}{\varepsilon}, \frac{1}{\delta}, n)$.*

8 Schwache PAC-Algorithmen

Wir erinnern zuerst an die Definition schwacher PAC-Algorithmen:

> L heißt <u>schwacher PAC-Algorithmus</u> für eine parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ (mit Namensfunktion $name : \mathcal{C} \rightarrow \{0,1\}^*$) falls es ein Polynom $p(n, l)$ gibt, so daß für alle δ, n, Verteilungen D und Zielkonzepte $c \in \mathcal{C}_n$ gilt:
>
> $$\mathrm{prob}[\mathrm{Fehler_D(c, h_c)} \leq \frac{1}{2} - \frac{1}{\mathrm{p(n,l)}}] \geq 1 - \delta,$$
>
> wobei h_c ist die von L ausgegebene Hypothese und $l = |name(c)|$ ist.

Ein schwacher PAC-Algorithmus liefert also eine Hypothese ab, die eine recht schwache Voraussagekraft hat. Wir konstruieren einen Boosting-Algorithmus, der jeden schwachen PAC-Algorithmus in einen (starken) PAC-Algorithmus verwandelt. Neben diesem Resultat ist die Konstruktion des Boosting-Algorithmus auch von allgemeinem Interesse: Wir werden sehen, daß es von Vorteil ist, wenn wir nicht eine sondern viele Hypothese bilden und die endgültige Hypothese durch Mehrheitsbildung bestimmen.

Wir beginnen mit der Grobstruktur des Algorithmus:
procedure Boosting

1. Berechne k als Funktion von ε, δ und n.
2. for $i := 0$ to $k-1$ do

   begin

   rufe das Programm $Example(i)$ auf, um Beispiele zu generieren;

   rufe den schwachen Lernalgorithmus L mit Parametern n und $\frac{\delta}{k}$ auf. L stellt die Hypothese h_i auf.

   end
3. Gib $h \equiv Mehrheit(h_0, \ldots, h_{k-1})$ als endgültige Hypothese aus.

Wir müssen also klären, wann Mehrheitsbildung zu einem verringerten Fehler führt. Dabei stellt sich heraus, daß die Prozedur „$Example$" eine zentrale Rolle spielt.

Wir beginnen mit einer Analyse der unvollständig spezifizierten Boosting-Prozedur. Die Analyse führt uns auf eine erfolgreiche Definition der Example-Prozedur.

Die Parameter n, δ und ε sowie die Verteilung D seien gegeben. Wir nehmen an, daß der schwache PAC-Algorithmus die Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ durch die Hypothesenklasse $\mathcal{H} = (\mathcal{H}_n \mid n \in \mathbb{N})$ erlernt. Das unbekannte Konzept $c \in \mathcal{C}_n$ sei zu erlernen.

Wir führen das Boosting-Spiel ein, das zwischen dem Booster und seinem Gegner gespielt wird. Dem Booster entspricht dabei die Example-Prozedur. Sein Gegner spielt die Rolle des schwachen Lernalgorithmus.

Die Spielregeln: Sei $\vartheta = \frac{1}{p(n,|name(c)|)}$.
Booster und Gegner wechseln sich ab und spielen k Runden. Der Booster beginnt in jeder Runde. In Runde i wählt der Booster eine Verteilung D_i auf dem Beispielsraum $\Sigma^{\leq n}$. Sein Gegner kontert mit einer Teilmenge $richtig_i \subseteq \Sigma^{\leq n}$, so daß $prob_{D_i}(richtig_i) \geq \frac{1}{2} + \vartheta$ gilt.

Wir modellieren mit dem Boosting-Spiel, daß die Example-Prozedur (bzw. der Booster) eine Verteilung wählt, während der schwache PAC-Algorithmus (der Kontrahent des Boosters) eine Hypothese h_i liefert, die auf der Menge $richtig_i$ richtig ist.

Die Spielauswertung:
Der Booster erhält die Menge

$$richtig = \{x \in \Sigma^{\leq n} \mid x \text{ wird mindestens } \lfloor \frac{\mathrm{k}}{2} \rfloor + 1 \text{ mal richtig klassifiziert}\}$$

als Belohnung. Der Booster hat das Spiel gewonnen, wenn

$$prob_D[richtig] \geq 1 - \varepsilon,$$

für einen Fehlerparameter ε gilt. Als nächstes beschreiben wir die Auswertung der Mehrheitshypothese des Boosting-Algorihtmus: $Mehrheit(h_1, \ldots, h_k)$ ist genau dann eine gute Hypothese, wenn $prob_D[richtig] \geq 1 - \varepsilon$ ist. Beachte, daß der Booster (bzw. die Example-Prozedur) Beispiele anfordern kann, die gemäß D generiert werden. Der Booster kennt aber das Zielkonzept c nicht. Wir beschreiben jetzt eine Gewinnstrategie für den Booster: Für i mit $0 \leq i \leq k$ sei

$$S_r^i = \{x \in \Sigma^{\leq n} \mid x \text{ wird in den ersten i Runden r mal richtig klassifiziert}\}.$$

Offensichtlich gilt für jedes $x \in S_r^i$, daß entweder $x \in S_r^{i+1}$ oder $x \in S_{r+1}^{i+1}$ gilt.

Die Strategie des Boosters in der **letzten Runde** ist klar: Er legt sein ganzes Gewicht auf Beispiele aus der Menge $S_{\lfloor \frac{k}{2} \rfloor}^{k-1}$, denn er hat $\bigcup\limits_{r > \lfloor \frac{k}{2} \rfloor} S_r^{k-1}$ als Belohnung schon in der Tasche, während $\bigcup\limits_{r < \lfloor \frac{k}{2} \rfloor} S_r^{k-1}$ für ihn verloren ist.

Was macht der Booster in der **vorletzten Runde**?

Nur das Schicksal von Elementen in $S_{\lfloor \frac{k}{2} \rfloor - 1}^{k-2} \cup S_{\lfloor \frac{k}{2} \rfloor}^{k-2}$ ist noch ungeklärt. Der Booster legt das Gewicht $\frac{1}{2} + \vartheta$ auf $S_{\lfloor \frac{k}{2} \rfloor - 1}^{k-2}$, während $S_{\lfloor \frac{k}{2} \rfloor}^{k-2}$ das Gewicht $\frac{1}{2} - \vartheta$ erhält. Der Grund warum auf $S_{\lfloor \frac{k}{2} \rfloor - 1}^{k-2}$ ein etwas größeres Gewicht gelegt wird ist klar: Der Booster kann sein Glück in der letzten Runde nochmals an der Menge $S_{\lfloor \frac{k}{2} \rfloor}^{k-1}$ versuchen und dort sein ganzes Gewicht investieren.

Allgemein gilt, daß der Booster in Runde $i+1$ der Menge S_r^i das Gewicht α_r^i (mit $\sum\limits_{r=0}^{i} \alpha_r^i = 1$) gibt. Innerhalb von S_r^i wählt der Booster die Verteilung D. Der Booster wählt also für $A \subseteq \{0,1\}^{\leq n}$ die Verteilung D_i mit

$$prob_{D_i}[A] = \frac{\sum\limits_{r=0}^{i} prob_D[A \cap S_r^i] \cdot \alpha_r^i}{Z},$$

wobei Z ein Normalisierungsfaktor mit $Z = \sum\limits_{r=0}^{i} prob_D[S_r^i] \cdot \alpha_r^i$ ist.

Wir definieren die Gewichte α_r^i gemäß der folgenden Strategie: Wir kümmern uns nicht um die Elemente, deren Schicksal schon entschieden ist:

$$\alpha_r^i = 0, \text{ falls } r + (k-i) < \lfloor \tfrac{k}{2} \rfloor \text{ oder } r > \lfloor \tfrac{k}{2} \rfloor. \qquad (*)$$

Wie oben besprochen wird

$$\alpha_{\lfloor \frac{k}{2} \rfloor}^{k-1} = 1$$

gesetzt, um in der letzten Runde das ganze Gewicht auf die Menge $S^{k-1}_{\lfloor\frac{k}{2}\rfloor}$ zu verteilen. Auf den unsicheren Elementen wird eine Binomialverteilung so gewählt, daß der erwartete Ausgang nach $k-1$ Runden bei $\lfloor\frac{k}{2}\rfloor$ liegt. Diese Elemente versuchen wir dann in der letzten Runde „aus dem Feuer zu reißen". Formal definieren wir für $i < k-1$:

$$\alpha^i_r = \text{ Wahrscheinlichkeit, daß in } k-1-i \text{ Versuchen } \lfloor\frac{k}{2}\rfloor - r \text{ Erfolge erzielt werden,}$$

wobei $\frac{1}{2}+\vartheta$ die Erfolgswahrscheinlichkeit ist.

8.1 Lemma Es gilt

a) $\alpha^{k-1}_r = \begin{cases} 1 & \text{wenn r} = \lfloor\frac{\text{k}}{2}\rfloor \\ 0 & \text{sonst} \end{cases}$

b) $\alpha^i_r = \left(\frac{1}{2}-\vartheta\right)\alpha^{i+1}_r + \left(\frac{1}{2}+\vartheta\right)\alpha^{i+1}_{r+1}$.

Beweis **a)** folgt direkt aus $(*)$.

b) Wenn in $k-i-1$ Versuchen $\lfloor\frac{k}{2}\rfloor - r$ Erfolge erzielt werden, dann wurden in $k-i-2$ Versuchen entweder $\lfloor\frac{k}{2}\rfloor - r$ oder $\lfloor\frac{k}{2}\rfloor-(r+1)$ Erfolge mit den entsprechenden Wahrscheinlichkeiten erzielt. □

Wir kommen jetzt zur Auswertung der Booster-Strategie. Wir definieren β^i_r als die Wahrscheinlichkeit, daß in $k-i$ Versuchen höchstens $\lfloor\frac{k}{2}\rfloor - r$ Erfolge, bei einer Erfolgswahrscheinlichkeit von $\left(\frac{1}{2}+\vartheta\right)$ pro Versuch, erzielt werden. Wir setzen

$$\beta^k_r = \begin{cases} 1 & r \leq \lfloor\frac{k}{2}\rfloor \\ 0 & \text{sonst.} \end{cases}$$

8.2 Lemma **a)** $\alpha^i_r = \beta^{i+1}_r - \beta^{i+1}_{r+1}$ für $0 \leq \text{i}, \text{r} < \text{k}$.

b) Für i mit $0 \leq i < k$ gilt die Rekursion

$$\beta^i_r = \left(\frac{1}{2}-\vartheta\right)\beta^{i+1}_r + \left(\frac{1}{2}+\vartheta\right)\beta^{i+1}_{r+1}.$$

Beweis **a)** Wenn in $k-i-1$ Versuchen genau $\lfloor\frac{k}{2}\rfloor - r$ Erfolge verzeichnet wurden, dann gab es höchstens $\lfloor\frac{k}{2}\rfloor - r$ Erfolge, aber mehr als $\lfloor\frac{k}{2}\rfloor - r - 1$ Erfolge.

b) Es genügt zu zeigen, daß

$$\beta^i_r = \left(\frac{1}{2}-\vartheta\right)\underbrace{(\beta^{i+1}_r - \beta^{i+1}_{r+1})}_{=\alpha^i_r} + \beta^{i+1}_{r+1}.$$

Warum ist dies richtig? Wenn in $k-i$ Versuchen höchstens $\lfloor\frac{k}{2}\rfloor - r$ Versuche erfolgreich waren, dann sind in $k-i-1$ Versuchen genau $\lfloor\frac{k}{2}\rfloor - r$ Versuche oder höchstens $\lfloor\frac{k}{2}\rfloor - r - 1$ Versuche mit den entsprechenden Wahrscheinlichkeiten erfolgreich. □

Warum ist die β-Folge von Relevanz? Es ist

$$\beta_0^0 = \text{ Wahrscheinlichkeit, daß in } k \text{ Versuchen höchstens } \lfloor \tfrac{k}{2} \rfloor \text{ Erfolge erzielt werden.}$$

Leider gilt die Beziehung $1 - \beta_0^0 = prob_D[richtig]$ nicht : Die Binomialverteilung wird im allgemeinen nicht mit der gegebenen Verteilung D übereinstimmen. Aber:

8.3 Lemma Es gilt $prob_D[richtig] \geq 1 - \beta_0^0$.

Beweis Wir weisen nach, daß

$$\sum_{r=0}^{i} \mathrm{prob_D(S_r^i)} \cdot \beta_\mathrm{r}^\mathrm{i} \geq \sum_{\mathrm{r=0}}^{\mathrm{i+1}} \mathrm{prob_D(S_r^{i+1})} \cdot \beta_\mathrm{r}^\mathrm{i+1}. \qquad (*)$$

Damit erhalten wir insbesondere, daß

$$\sum_{r=0}^{0} \mathrm{prob_D(S_0^0)} \cdot \beta_0^0 \geq \sum_{\mathrm{r=0}}^{\mathrm{k}} \mathrm{prob_D(S_r^k)} \cdot \beta_\mathrm{r}^\mathrm{k}$$

gilt. Da $S_0^0 = \Sigma^{\leq n}$, ist $\mathrm{prob}_D(S_0^0) = 1$. Andererseits ist

$$\beta_r^k = \begin{cases} 1 & r \leq \lfloor \frac{k}{2} \rfloor \\ 0 & \text{sonst} \end{cases}$$

und wir erhalten

$$\begin{aligned} \beta_0^0 &\geq \sum_{r=0}^{\lfloor \frac{k}{2} \rfloor} prob_D[S_r^k] \\ &= 1 - prob_D[richtig]. \end{aligned}$$

Um $(*)$ herzuleiten, stellen wir zuerst einen Zusammenhang zwischen S_r^i und S_{r+1}^{i+1} her. Sei $M_r^i = S_r^i \cap S_{r+1}^{i+1}$ und $x_r^i = \frac{prob_D[M_r^i]}{prob_D[S_r^i]}$. (Insbesondere ist also $x_r^i \cdot prob_D[S_r^i] = prob_D[M_r^i]$.) Dann gilt:

$$\begin{aligned} prob_D[S_0^{i+1}] &= prob_D[S_0^i](1 - x_0^i), \\ prob_D[S_r^{i+1}] &= prob_D[S_{r-1}^i]x_{r-1}^i + prob_D[S_r^i](1 - x_r^i), \\ prob_D[S_{i+1}^{i+1}] &= prob_D[S_i^i] \cdot x_i^i. \end{aligned}$$

Damit ist

$$\begin{aligned} &\sum_{r=0}^{i+1} prob_D[S_r^{i+1}]\beta_r^{i+1} \\ &= prob_D[S_0^i](1 - x_0^i)\beta_0^{i+1} + \sum_{r=1}^{i} \left(prob_D[S_{r-1}^i]x_{r-1}^i + prob_D[S_r^i](1 - x_r^i) \right) \beta_r^{i+1} \\ &\qquad + prob_D[S_i^i]x_i^i \cdot \beta_{i+1}^{i+1} \end{aligned}$$

$$= \sum_{r=0}^{i} prob_D[S_r^i]\left((1-x_r^i)\beta_r^{i+1} + x_r^i \cdot \beta_{r+1}^{i+1}\right)$$
$$= \sum_{r=0}^{i} prob_D[S_r^i]\beta_r^{i+1} + \sum_{r=0}^{i} prob_D[S_r^i]x_r^i(\beta_{r+1}^{i+1} - \beta_r^{i+1}).$$

Wir betrachten den zweiten Summanden. Da

$$prob_D[S_r^i]x_r^i = prob_D[M_r^i]$$

und

$$\beta_r^{i+1} - \beta_{r+1}^{i+1} = \alpha_r^i,$$

gilt:

$$\sum_{r=0}^{i} prob_D[S_r^i]x_r^i(\beta_{r+1}^{i+1} - \beta_r^{i+1}) = -\sum_{r=0}^{i} prob_D[M_r^i]\alpha_r^i.$$

Die Menge $\bigcup_{r=0}^{i} M_r^i$ ist die Belohnung für den Booster in der $(i+1)$-ten Runde. Nach den Spielregeln muß also gelten

$$\frac{1}{Z}\sum_{r=0}^{i} prob_D[M_r^i]\alpha_r^i \geq \frac{1}{2} + \vartheta.$$

Da $Z = \sum_{r=0}^{i} prob_D[S_r^i]\alpha_r^i$ ist, gilt

$$\sum_{r=0}^{i} prob_D[S_r^i]x_r^i\alpha_r^i \geq \left(\frac{1}{2} + \vartheta\right)\sum_{r=0}^{i} prob_D[S_r^i]\alpha_r^i.$$

Deshalb ist (mit $\alpha_r^i = \beta_r^{i+1} - \beta_{r+1}^{i+1}$)

$$\begin{aligned}
\sum_{r=0}^{i+1} prob_D[S_r^{i+1}]\beta_r^{i+1} &\leq \sum_{r=0}^{i} prob_D[S_r^i]\beta_r^{i+1} + \left(\frac{1}{2} + \vartheta\right)\sum_{r=0}^{i} prob_D[S_r^i](\beta_{r+1}^{i+1} - \beta_r^{i+1}) \\
&= \sum_{r=0}^{i} prob_D[S_r^i]\left(\left(\frac{1}{2} + \vartheta\right)\beta_{r+1}^{i+1} + \left(\frac{1}{2} - \vartheta\right)\beta_r^{i+1}\right) \\
&= \sum_{r=0}^{i} prob_D[S_r^i]\beta_r^i,
\end{aligned}$$

und dies war zu zeigen. □

Die Strategie des Boosters liefert nach nicht zu vielen Runden die gewünschte Belohnung $1-\varepsilon$:

8.4 Lemma Wenn $k \geq 2\left(\frac{1}{\vartheta}\right)^2 \ln\left(\frac{1}{\varepsilon}\right)$ gilt, dann ist $prob_D[richtig] \geq 1 - \varepsilon$.

Beweis Mit Lemma 8.3 gilt $prob_D[richtig] \geq 1 - \beta_0^0$ und

$$\beta_0^0 = prob\left[\sum_{i=1}^{k} X_i \leq \lfloor\frac{k}{2}\rfloor\right].$$

Dabei ist $prob[X_i = 1] = \frac{1}{2} + \vartheta$ und die erwartete Anzahl von Erfolgen ist $k\left(\frac{1}{2} + \vartheta\right)$. Es ist $(1-\alpha)k\left(\frac{1}{2} + \vartheta\right) = \frac{k}{2}$ für $\alpha = \frac{2\vartheta}{1+2\vartheta}$ und damit liefert die Chernoff-Schranke

$$\beta_0^0 \leq e^{-\frac{\alpha^2 k(\frac{1}{2}+\vartheta)}{2}}.$$

Die Forderung $\beta_0^0 \leq \varepsilon$ ist erfüllt, falls

$$\begin{aligned} -\frac{\alpha^2 k\ (\frac{1}{2}+\vartheta)}{2} &= -\frac{4\vartheta^2}{2} k\ \frac{1/2+\vartheta}{(1+2\vartheta)^2} \\ &= -\frac{4\vartheta^2}{2} k\ \frac{1}{2(1+2\vartheta)} \\ &= -\vartheta^2 \cdot k \cdot \frac{1}{1+2\vartheta} \\ &\overset{!}{\leq} \ln(\varepsilon) \end{aligned}$$

erfüllt ist. Dementsprechend verlangen wir

$$k \overset{!}{\geq} \ln\left(\frac{1}{\varepsilon}\right) \frac{1+2\vartheta}{\vartheta^2}$$

und diese Ungleichung ist erfüllt, wenn

$$k = \ln\left(\frac{1}{\varepsilon}\right) \cdot \left(\frac{1}{\vartheta}\right)^2$$

gilt, da $2 \geq 1 + 2\vartheta$. □

Wir können jetzt die Beschreibung der Boosting-Prodezur vervollständigen:

- Setze $k = 2 \cdot \left(\frac{1}{\vartheta}\right)^2 \ln\left(\frac{1}{\varepsilon}\right)$.
- Die Prozedur $Example(i)$ hat die folgende Form:

  Wiederhole bis genügend viele Beispiele vorhanden sind
  1. Fordere ein Beispiel x an.
  2. Berechne $h_0(x), \ldots, h_{i-1}(x)$ und $r = |\{j \mid h_j \text{ klassifiziert } x \text{ richtig }\}|$.
  3. Mit Wahrscheinlichkeit α_r^i akzeptiere x als ein Beispiel.

Beachte, daß die Prozedur $Example(i)$ genau die Verteilung D_i produziert. Wir können jetzt zusammenfassen:

8.5 Satz *Sei L ein schwacher PAC-Algorithmus, der polynomielle Zeit in den Parametern $\frac{1}{\delta}$, n und $\min\{|w| \mid w \in name(c)\}$ benötigt. Dann produziert der Boosting-Algorithmus einen polynomiellen PAC-Algorithmus.*

Beweis Wenn wir die Laufzeit der Example-Prozedur ausklammern, dann erhöht sich die Laufeit um den Faktor $k = poly\left(\frac{1}{\vartheta}, \ln\left(\frac{1}{\varepsilon}\right)\right)$.

Die Example-Prozedur muß die vom schwachen Lernalgorithmus gewünschte Anzahl von Beispielen liefern. Wenn mindestens eine Rate von $\left(\frac{1}{\varepsilon}\right)^2$ aller Beispiele akzeptiert wird, dann läuft Example auch in Zeit $poly\left(\frac{1}{\varepsilon}, \frac{1}{\vartheta}, n, \min\{|w| \mid w \in name(c)\}\right)$.

Was tun, wenn die Akzeptanzrate $\left(\frac{1}{\varepsilon}\right)^2$ unterschreitet?
In diesem Fall ist das Schicksal der überwiegenden Mehrheit aller Beispiele schon besiegelt: Die Mehrheit der bisher erarbeiteten Hypothesen ist bereits gut genug. (Diese Behauptung werden wir nicht weiter beweisen.)

Im Boosting-Spiel gehen wir davon aus, daß der Gegner (also der Lernalgorithmus) **stets** eine gute Hypothese findet. Diese Annahme ist legal, da wir in der Boosting-Prozedur $\frac{\delta}{k}$ als Vertrauensparameter für den schwachen PAC-Algorithmus L wählen: Mit Wahrscheinlichkeit δ sind **alle** Hypothesen gut. □

9 Übungsaufgaben

Blatt 1

Aufgabe 1.1

Ein Lernalgorithmus für das PAC-Modell fordert zuerst eine Menge von klassifizierten Beispielen an. Dann wird eine Hypothese h gesucht, die konsistent mit allen gesehenen Beispielen ist.
(Für die Beispielmenge $B = \{(x_1, b_1), \ldots, (x_m, b_m)\}$, mit $x_1, \ldots, x_m \in \Sigma^*$ und $b_1, \ldots, b_m \in \{0, 1\}$, heißt eine Hypothese $h \subseteq \Sigma^*$ genau dann konsistent, wenn $(x_i \in h \Leftrightarrow b_i = 1)$ für $1 \leq i \leq m$ gilt.)

Zeige: Für jede Menge $B \subseteq \{0,1\}^{n+1}$ von m Beispielen kann eine konsistente $k - CNF$ Formel über n Variablen in polynomieller Zeit (in m und n^k) gefunden werden, falls eine solche Formel existiert.

Damit kann das Konsistenzproblem für die Konzeptklasse der k-CNF Formeln mit k-CNF Formeln als Hypothesenklasse effizient gelöst werden.

Aufgabe 1.2

In dieser Aufgabe zeigen wir die Wichtigkeit der Wahl der Hypothesenklasse. In der Vorlesung werden wir zeigen, daß das Konsistenzproblem für k-Term DNF Formeln nicht effizient gelöst werden kann, falls k-Term DNF Formeln auch als Hypothesenklasse verwendet werden. Diese Aussage gilt, falls $NP \neq P$.

Zeige: Zu jeder k-Term DNF Formel α gibt es eine äquivalente k-CNF Formel β.

(Das Konsistenzproblem für k-Term DNF Formeln läßt sich also mit Aufgabe 1.1 effizient lösen, wenn k-CNF Formeln als Hypothesenklasse benutzt werden.)

Aufgabe 1.3

Sei $\mathcal{C}$ eine Konzeptklasse über Σ. Wir sagen, daß $\mathcal{C}$ eine Menge $S \subseteq \Sigma^*$ zertrümmert, falls

$$\{c \cap S \mid c \in \mathcal{C}\} = \mathcal{P}(S)$$

gilt, wobei $\mathcal{P}(S)$ die Potenzmenge von S ist.

Die Vapnik-Chervonenkis Dimension von $\mathcal{C}$ (Notation: $VC(\mathcal{C})$) ist definiert als die maximale Größe einer Menge S, die von $\mathcal{C}$ zertrümmert wird.
Zeige:

a) $2^{VC(\mathcal{C})} \leq |\mathcal{C}|$ und

b) $VC(Monom_n) = \Theta(n)$, wobei die Konzeptklasse $Monom_n$ alle Monome über den $2n$ Literalen $x_1, \neg x_1, \ldots, x_n, \neg x_n$ beinhaltet.

Blatt 2

Aufgabe 2.1

a) Seien $\mathcal{C}_1$ und $\mathcal{C}_2$ Konzeptklassen. **Zeige**, daß folgendes gilt:

$$\mathcal{C}_1 \subseteq \mathcal{C}_2 \Rightarrow VC(\mathcal{C}_1) \leq VC(\mathcal{C}_2).$$

b) Für die Konzeptklasse $\mathcal{C}$ sei $\overline{\mathcal{C}} = \{\bar{c} \mid c \in \mathcal{C}\}$. **Zeige**, daß folgendes gilt:

$$VC(\overline{\mathcal{C}}) = VC(\mathcal{C}).$$

Aufgabe 2.2

Zeige: $VC(Sym_n) = n + 1$.

Aufgabe 2.3

Ein Monom m heißt monoton, wenn m eine Konjunktion von **un**negierten Literalen ist. Die Konzeptklasse $Monoton\text{-}Monom_n$ enthält für jedes monotone Monom m die Menge

$$wahr(m) = \{x \mid x \in \{0,1\}^n,\ x \text{ erfüllt } m\ \}$$

als Konzept.
Zeige: $VC(Monoton\text{-}Monom_n) = n$.

Aufgabe 2.4

a) **Zeige:** $VC(k\text{-}DNF_n) \leq \binom{2n}{k} + \binom{2n}{k-1} + \cdots + \binom{2n}{1} + 1$.
Bessere Abschätzungen existieren und sind als Lösungen willkommen.

b) **Zeige:** $VC(k\text{-}DNF_n) \geq \binom{n}{k}$.

c) **Zeige:** $VC(k\text{-}CNF_n) = VC(k\text{-}DNF_n)$.

Kommentar: Die Konzeptklasse $k\text{-}CNF$ ist für jedes feste k effizient lernbar: Nach Aufgabe 1.1 können wir das Konsistenzproblem effizient lösen. In der Vorlesung werden wir sehen, daß eine Konzeptklasse der VC-Dimension d sich mit $O(\frac{1}{\varepsilon}(n \cdot d + \ln(\frac{1}{\delta})))$ Beispielen im PAC-Modell erlernen läßt. Damit reicht eine polynomielle Anzahl (in $\frac{1}{\varepsilon}, n^k$ und $\frac{1}{\delta}$) von Beispielen für $k\text{-}CNF_n$ aus.

Blatt 3

Aufgabe 3.1

a) **Zeige:** $VC(Rechteck_{n,d}) = 2d$.

b) Entwirf einen PAC-Algorithmus, der $Rechteck_{n,d}$ in polynomieller Zeit erlernt. Welche Namensfunktion für $Rechteck_{n,d}$ wird dabei verwendet?

Aufgabe 3.2

Zeige: Für $k \leq 2^{n/2}$ gilt: $VC(k\text{-Term } DNF_n) = \Theta(n \cdot k)$.

Kommentar: Für konstantes k gilt: $VC(k\text{-}CNF_n) = \Theta(n^k)$ und $VC(k\text{-Term } DNF_n) = \Theta(n)$. Also führt das Lernen von k-Term DNF_n durch k-CNF_n zu einer sehr viel größeren Anzahl an angeforderten Beispielen.

Aufgabe 3.3

In dieser Aufgabe besprechen wir PAC-Lernalgorithmen mit einseitigem Fehler.
Ein Lernalgorithmus erlernt eine Konzeptklasse $\mathcal{C}$ mit einseitigem Fehler im PAC-Sinne, wenn für jedes ε, δ und n, für jedes Konzept $c \in \mathcal{C}$ und für jede Verteilung D auf $\Sigma^{\leq n}$ gilt:

- $h \subseteq c$ und
- $prob[Fehler_D(c,h) \leq \varepsilon] \geq 1 - \delta$,

wobei h die vom Algorithmus gefundene Hypothese ist. Im folgenden versuchen wir, Bedingungen zu finden, die garantieren, daß $\mathcal{C}$ mit einseitigem Fehler im PAC-Sinne erlernbar ist.

Eine Konzeptklasse $\mathcal{C}$ heißt minimal konsistent, wenn es zu jeder endlichen Menge von klassifizierten Beispielen $S \subseteq \Sigma^*$ ein (mit S konsistentes) Konzept h gibt, so daß h in **jedem** mit S konsistenten Konzept enthalten ist.

a) Sei $\mathcal{C}$ eine minimal konsistente Konzeptklasse.
Zeige: Dann gibt es einen PAC-Algorithmus, der $\mathcal{C}$ mit einseitigem Fehler lernt und höchstens $\frac{1}{\varepsilon}(\ln(|\mathcal{C}|) + \ln(\frac{1}{\delta}))$ Beispiele anfordert. (Wir stellen hier keine Anforderungen an die Effizienz des PAC-Algorithmus!)

b) **Zeige:** Wenn es einen PAC-Algorithmus für $\mathcal{C}$ gibt, der $\mathcal{C}$ mit einseitigem Fehler erlernt, dann ist $\mathcal{C}$ minimal konsistent.

Achtung: Ein PAC-Algorithmus mit einseitigem Fehler muß noch nicht einmal konsistent auf der Beispielmenge sein! Aber wir können den Algorithmus durch geeignete Wahl von ε, δ und der Verteilung D zur Konsistenz zwingen. Wie und warum?

c) **Zeige,** ob die Konzeptklasse $Monom_n$ und, ob die Konzeptklasse k-DNF_n minimal konsistent ist oder nicht.

Blatt 4

Aufgabe 4.1

Ein strikt binärer Baum ist ein geordneter Baum, dessen innere Knoten stets genau zwei Kinder besitzen.

Ein Entscheidungsbaum über den Booleschen Variablen $x_1, \dots, x_n$ ist ein strikt binärer Baum, dessen innere Knoten mit einem Element aus $\{x_1, \dots, x_n\}$ und dessen Blätter mit den Wahrheitswerten 0 oder 1 markiert sind. Eine Eingabe $y = (y_1, \dots, y_n) \in \{0,1\}^n$ wird an der Wurzel angelegt. Erreicht die Eingabe den Knoten v mit Markierung x_i wird sie zum linken (rechten) Kind weitergeleitet, falls $y_i = 0$ ($y_i = 1$) ist. Eine Eingabe erfüllt den Baum, wenn sie ein mit 1 markiertes Blatt erreicht.

$Decision_{n,s}$ besitzt für jeden Entscheidungsbaum T über den Booleschen Variablen $x_1, \dots,$ x_n das Konzept

$$wahr(T) = \{x \mid x \in \{0,1\}^n, x \text{ erfüllt } T\}.$$

Dabei setzen wir voraus, daß T höchstens s Knoten besitzt.

a) Bestimme eine möglichst gute obere Schranke für die Anzahl der strikt binären Bäumen mit höchstens p Knoten.

b) **Zeige**, daß $Beispiel(Decision_{n,p}) = O(\frac{1}{\varepsilon}(p \cdot \ln(n) + \ln(\frac{1}{\delta})))$ gilt.

Aufgabe 4.2

Sei $\mathcal{C}$ eine Konzeptklasse über Σ. Für $X \subseteq \Sigma^*$ setzen wir:

$$v_{\mathcal{C}}(X) = |\{X \cap c \mid c \in \mathcal{C}\}|$$

und definieren für jedes $m \in \mathbb{N}$:

$$v_{\mathcal{C}}(m) = \max\{v_{\mathcal{C}}(X) \mid |X| = m\}.$$

a) **Zeige:** Für $m \leq VC(\mathcal{C})$ gilt $v_{\mathcal{C}}(m) = 2^m$ und
für $m > VC(\mathcal{C})$ gilt $v_{\mathcal{C}}(m) < 2^m$.

b) **Zeige**, daß für $m \geq d$ $v_{\mathcal{C}}(m) \leq \Phi(m,d) \leq \left(\frac{e \cdot m}{d}\right)^d$ gilt, wobei $d = VC(\mathcal{C})$ und $\Phi(m,d) = \sum_{i=0}^{d} \binom{m}{i}$ ist.

Hinweis zu b): Wende Lemma 2.4 an, um zu zeigen, daß $v_{\mathcal{C}}(X) \leq \sum_{i=0}^{d} \binom{|X|}{i}$ für alle $X \subseteq \Sigma^*$ (mit $|X| = m$) gilt.

Aufgabe 4.3

a) $\mathcal{C}_1$ und $\mathcal{C}_2$ seien zwei Konzeptklassen über Σ. Wir definieren die Konzeptklasse $\mathcal{C}_1 \vee \mathcal{C}_2$ durch

$$\mathcal{C}_1 \vee \mathcal{C}_2 = \{c_1 \cup c_2 \mid c_1 \in \mathcal{C}_1,\ c_2 \in \mathcal{C}_2\}.$$

Zeige, daß $v_{\mathcal{C}_1 \vee \mathcal{C}_2}(m) \leq v_{\mathcal{C}_1}(m) \cdot v_{\mathcal{C}_2}(m)$ für jedes $m \in \mathbb{N}$ gilt.

b) Sei $\mathcal{C}$ eine Konzeptklasse über Σ, mit $VC(\mathcal{C}) = d$, $d \geq 1$. Wir definieren die Konzeptklasse k-Term $\mathcal{C} = \underbrace{\mathcal{C} \vee \mathcal{C} \vee \ldots \vee \mathcal{C}}_{k\text{-mal}}$ durch

$$k\text{-Term } \mathcal{C} = \{c_1 \cup c_2 \cup \ldots \cup c_k \mid c_1, c_2, \ldots, c_k \in \mathcal{C}\}.$$

Zeige (durch eine entsprechende Verallgemeinerung von Aufgabe 4.3 a) und mit Hilfe von Aufgabe 4.2 b)), daß für $m \geq d$ die Ungleichung

$$v_{k\text{-Term } \mathcal{C}}(m) \leq \left(\frac{e \cdot m}{VC(\mathcal{C})}\right)^{k \cdot VC(\mathcal{C})}$$

gilt. **Zeige** dann mittels Aufgabe 4.2 a), daß

$$VC(k\text{-Term } \mathcal{C}) < 2 \cdot k \cdot VC(\mathcal{C}) \cdot \log_2(4k)$$

gilt.

<u>Kommentar:</u> Damit können wir von der VC Dimension einer Konzeptklasse $\mathcal{C}$ auf die VC Dimension der Konzeptklasse k-Term $\mathcal{C}$ schließen. Man beachte aber, daß unser Resultat nicht optimal sein muß. So erhalten wir zum Beispiel $VC(k\text{-Term } DNF_n) \leq 2 \cdot k \cdot VC(Monom_n) \cdot \log_2(4k)$, während $VC(k\text{-Term } DNF_n) = O(k \cdot VC(Monom_n))$ in der Vorlesung gezeigt wurde.

Analoge Resultate erhalten wir für $\mathcal{C}_1 \wedge \mathcal{C}_2 = \{c_1 \cap c_2 \mid c_1 \in \mathcal{C}_1,\ c_2 \in \mathcal{C}_2\}$ bzw. für $\underbrace{\mathcal{C} \wedge \ldots \wedge \mathcal{C}}_{k\text{-mal}} = \{c_1 \cap \ldots \cap c_k \mid c_1, \ldots, c_k \in \mathcal{C}\}$.

Blatt 5

Aufgabe 5.1

Gib möglichst gute untere und obere Schranken für die VC-Dimension der folgenden **reellwertigen** Konzeptklassen an:

a) $\mathcal{C}_1$ besitzt für jeden Kreis K in $\mathbb{R}^2$ ein Konzept. Für den Kreis mit Mittelpunkt x und Radius r besteht das entsprechende Konzept aus allen Punkten $y \in \mathbb{R}^2$ mit $||x-y|| \leq r$.

b) Sei k fest gewählt. $\mathcal{C}_2$ besitzt für jedes konvexe Polygon P in $\mathbb{R}^2$ mit k Kanten ein Konzept. Dieses Konzept besteht aus allen Punkten des $\mathbb{R}^2$, die zum Inneren oder zum Rand von P gehören.

Aufgabe 5.2

Eine k-Entscheidungsliste für Eingabelänge n ist durch einen Vektor der Form $((c_1, b_1), (c_2, b_2), \ldots, (c_s, b_s), b_{s+1})$ spezifiziert. Jedes c_i, $1 \leq i \leq s$, ist hierbei eine Konjunktion von höchstens k Literalen aus $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Die b_i, $1 \leq i \leq s+1$, sind Wahrheitswerte aus der Menge $\{0, 1\}$. Für eine Eingabe $y = (y_1, \ldots, y_n) \in \{0,1\}^n$ wird das kleinste i mit $c_i(y) = 1$ bestimmt und dann die Ausgabe b_i gegeben. Falls $c_i(y) = 0$ für alle i, $1 \leq i \leq s$, ist, wird die Ausgabe b_{s+1} gegeben.

Die Konzeptklasse $k\text{-}Liste_n$ besitzt für jede k-Entscheidungsliste der Eingabelänge n die entsprechende Menge der erfüllenden Wahrheitsbelegungen als ein Konzept.

Sei k konstant.

a) **Zeige:** $k\text{-}DNF_n \subseteq k\text{-}Liste_n$ und $k\text{-}CNF_n \subseteq k\text{-}Liste_n$.

b) **Bestimme** eine möglichst gute untere und eine möglichst gute obere Schranke für

$$Beispiel(k\text{-}Liste_n).$$

c) **Konstruiere** einen polynomiellen PAC-Algorithmus für $k\text{-}Liste_n$.

Aufgabe 5.3

Ein Äquivalenz-Lernalgorithmus für eine parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ besitzt die folgende Form:

```
Stelle eine Anfangshypothese h auf;
while (h nicht korrekt) do
   begin
      fordere ein Gegenbeispiel an;
      modifiziere h so, daß h mit allen bisher erhaltenen Gegenbeispielen konsistent ist;
   end.
```

Sei A ein Äquivalenz-Lernalgorithmus, der jedes Konzept in $\mathcal{C}_n$ nach höchstens $poly_1(n)$ vielen Gegenbeispielen **exakt** bestimmt.

Das Ziel dieser Aufgabe ist die Konstruktion eines polynomiellen PAC-Algorithmus L für $\mathcal{C}$. Aus der Existenz von polynomiellen Äquivalenz-Lernalgorithmen folgt somit die Existenz von polynomiellen PAC-Algorithmen. Die Umkehrung dieser Aussage gilt nicht.

Wir beschreiben L:

> L erhält ε, δ und n als Eingabe. L durchläuft höchstens $poly(n) + 1$ Phasen. In Phase 0 wird ein beliebiges Konzept als Hypothese h_0 aufgestellt. In Phase i werden
>
> $$\lceil \frac{1}{\varepsilon}(\ln(2)i + \ln(\frac{1}{\delta}))\rceil =: r_i$$
>
> klassifizierte Beispiele angefordert. Wenn alle Beispiele mit h_{i-1} konsistent sind, dann hält L mit der Hypothese h_{i-1}. Sonst wird A für jedes Gegenbeispiel aufgerufen, um die gegenwärtige Hypothese zu modifizieren. h_i ist die von L aufgestellte Hypothese nach Abarbeitung aller r_i Beispiele.

Zeige, daß L ein polynomieller PAC-Algorithmus ist. Wir setzen dabei voraus, daß es einen Algorithmus gibt, der für jedes Konzept $c \in \mathcal{C}_n$ und für jedes $x \in \Sigma^{\leq n}$ in Zeit höchstens $poly_2(n)$ entscheidet, ob $x \in c$ ist.

Blatt 6

In den Aufgaben 6.1 und 6.2 benötigen wir die folgende Binärkodierung natürlicher Zahlen:
Für $x \in \mathbb{N}$ mit $x = \sum_{i=0}^{k} \alpha_i 2^i$ (wobei $\alpha_k = 1$ ist) und $k = \sum_{i=0}^{r} \beta_i 2^i$ (wobei $\beta_r = 1$ ist) setze

$$Bin(x) = \beta_0\ \beta_0\ \beta_1\ \beta_1 \ldots \beta_r\ \beta_r\ 0\ 1\ \alpha_0\ \alpha_1 \ldots \alpha_k.$$

Diese Kodierung erlaubt es, Folgen von natürlichen Zahlen $(x_1, \ldots, x_t)$ effizient zu kodieren, nämlich durch das Wort $Bin(x_1) \ldots Bin(x_t)$. Eine Dekodierung ist stets möglich: Suche die erste Bitfolge $b_{2i-1}b_{2i} = 01$ und bestimme dann die Anzahl der Bits der ersten Zahl. Damit erhalten wir x_1. Wir wiederholen dieses Verfahren für die folgenden Zahlen.

Aufgabe 6.1

Ein Intervall $I = [a, b]$ über $\mathbb{N}$ besteht aus allen $x \in \mathbb{N}$ mit $a \leq x \leq b$. Wir nehmen stets an, daß die Endpunkte a und b natürliche Zahlen sind.

Wir betrachten die Konzeptklasse *Intervall*. *Intervall* besitzt für jede endliche Vereinigung $I_1 \cup \ldots \cup I_s$ von Intervallen über $\mathbb{N}$ ein Konzept $c_{I_1,\ldots,I_s}$. Das Konzept $c_{I_1,\ldots,I_s}$ besteht aus allen natürlichen Zahlen, die zu der Vereinigung $I_1 \cup \ldots \cup I_s$ gehören. (Z.B. Für $I_1 = [2,5]$, $I_2 = [9,9]$ ist $c_{I_1,I_2} = \{2,3,4,5,9\}$.)

Sei $I_j = [a_j, b_j]$, für $1 \leq j \leq s$. Die obige Binärkodierung von $(a_1, b_1, \ldots, a_s, b_s)$ dient als Namen für das Konzept $c_{I_1,\ldots,I_s}$. Das leere Konzept erhält den Namen $Bin(1) \cdot Bin(0)$.

Entwirf einen polynomiellen Occam-Algorithmus für *Intervall*, der eine konsistente Hypothese mit *kürzestem* Namen konstruiert.

Aufgabe 6.2

Ein Rechteck $R = [(x_{min}, y_{min}), (x_{max}, y_{max})]$ über $\mathbb{N}^2$ (mit $(x_{min}, y_{min}), (x_{max}, y_{max}) \in \mathbb{N}^2$) besteht aus allen Paaren $(x, y) \in \mathbb{N}^2$ mit $x_{min} \leq x \leq x_{max}$ und $y_{min} \leq y \leq y_{max}$.

Wir betrachten die Konzeptklasse *Rechteck*. *Rechteck* besitzt für jede endliche Vereinigung $R_1 \cup \cdots \cup R_s$ von Rechtecken über $\mathbb{N}^2$ das Konzept

$$c_{R_1,\ldots,R_s} = \{(x,y) \in \mathbb{N}^2 \mid \text{ es gibt } i, 1 \leq i \leq s, \text{ so daß } (x,y) \in R_i\}.$$

Sei $R_i = [(x_{min_i}, y_{min_i}), (x_{max_i}, y_{max_i})]$, für $1 \leq i \leq s$. Wir benutzen die obige Binärkodierung von $(x_{min_1}, y_{min_1}, x_{max_1}, y_{max_1}, \ldots, x_{min_s}, y_{min_s}, x_{max_s}, y_{max_s})$ als Namen für das Konzept $c_{R_1,\ldots,R_s}$.

Entwirf einen polynomiellen Occam-Algorithmus für *Rechteck*.

Hinweis: Das folgende Ergebnis darf benutzt werden: Seien ein Universum U sowie Mengen $S_1, \ldots, S_r \subseteq U$, mit $S_1 \cup \ldots \cup S_r = U$, gegeben. Dann gibt es einen Algorithmus

A, der eine Überdeckung von U mit höchstens $opt \cdot \ln(|U|) + 1$ Mengen aus $\{S_1, \ldots, S_r\}$ konstruiert. Dabei ist opt die kleinste Anzahl von Mengen aus $\{S_1, \ldots, S_r\}$, die für eine Überdeckung nötig ist. A läuft in Zeit $poly(r, |U|)$.

Aufgabe 6.3

Seien $\mathcal{C}$ und $\mathcal{H}$ Konzeptklassen. Wir betrachten fehlertolerante Lernalgorithmen, die möglicherweise falsch klassifizierte Beispiele erhalten. In der Praxis tritt dieser Fall bei ungenauen Beobachtungen auf.

Für $\beta \in [0, 1]$ nehmen wir dabei an, daß ein Beispiel mit Wahrscheinlichkeit mindestens $1 - \beta$ richtig klassifiziert wird, und die Klassifikation des Beispiels mit Wahrscheinlickeit β unvoraussagbar ist.

Ein Lernalgorithmus L heißt genau dann β-fehlertolerant, wenn für alle ε ($0 < \varepsilon \leq 1$, mit $\beta \leq \frac{\varepsilon}{4}$), für alle δ ($0 < \delta \leq 1$), für alle $n \in \mathbb{N}$, für alle Verteilungen D auf $\Sigma^{\leq n}$ und für alle Konzepte $c \in \mathcal{C}$

$$prob[Fehler_D(c, h_c) \leq \varepsilon] \geq 1 - \delta$$

gilt, wobei $h_c \in \mathcal{H}$ die von L aufgestellte Hypothese ist. (Die Beispiele werden gemäß der unbekannten Verteilung D generiert. Für die Klassifikation der Beispiele benutzen wir das obige Fehlermodell mit Parameter β.)

Ein Algorithmus L heißt β-konsistent, wenn für alle ε (mit $\beta \leq \frac{\varepsilon}{4}$), δ, n, D und $c \in \mathcal{C}_n$ mit Wahrscheinlichkeit mindestens $1 - \frac{\delta}{2}$ eine Hypothese h_c gefunden wird, die für höchstens $\frac{\varepsilon \cdot s}{2}$ der s angeforderten Beispiele nicht konsistent ist.

Bestimme s (als Funktion von ε, δ und $|\mathcal{H}|$), so daß ein β-konsistenter Lernalgorithmus ein β-fehlertoleranter Lernalgorithmus ist.

Mit anderen Worten: „Fast-konsistente" Hypothesen garantieren die PAC-Eigenschaft selbst bei fehlerhaften Beispielen, solange die Fehlerrate β nicht zu groß ist ($\beta \leq \varepsilon/4$).

Blatt 7

Ein Manhattan-Kurvenzug M (in $\mathbb{N} \times \mathbb{N}$) ist gegeben durch eine Folge $(x_1, y_1, x_2, y_2, \ldots, x_s, y_s)$ von Eckpunkten, mit $(x_i, y_i) \in \mathbb{N}^2$ $(1 \leq i \leq s)$. Dabei wird gefordert, daß (x_i, y_i) und (x_{i+1}, y_{i+1}), für $1 \leq i \leq s-1$, abwechselnd entweder in der x-Koordinate oder in der y-Koordinate übereinstimmen. M besteht aus allen Punkten in $\mathbb{N}^2$, die auf der (x_i, y_i) und (x_{i+1}, y_{i+1}) verbindenden Geraden liegen (für ein i mit $1 \leq i \leq s-1$). (Mit anderen Worten, die Kanten des Kurvenzugs verlaufen entweder parallel zur x-Achse oder parallel zur y-Achse.)

Aufgabe 7.1

Ein Manhatten-Kurvenzug M heißt x-monoton, wenn die x-Koordinaten der gegebenen Folge von Eckpunkten monoton steigend sind. (D.h. für jedes i mit $1 \leq i \leq s-1$ ist $x_i \leq x_{i+1}$.)

Setze $Bereich = \{0, 1, \ldots, 2^n - 1\} \times \{0, 1, \ldots 2^n - 1\}$. Sei $\mathcal{C}_n$ die Konzeptklasse, die für jeden x-monotonen Manhattan-Kurvenzug M mit Eckpunkten aus $Bereich$ (und somit $M \subseteq Bereich$) ein Konzept c_M besitzt, nämlich die Menge aller Punkte (x, y) aus M.

Der Name von c_M ist

$$bin_n(x_1) bin_n(y_1) bin_n(x_2) bin_n(y_2) \cdots bin_n(x_s) bin_n(y_s),$$

wobei $bin_n(u)$ die Binärdarstellung von u ist, die durch führende Nullen auf Länge n gebracht wird.

Die parametrisierte Konzeptklasse $\mathcal{C} = (\mathcal{C}_n \mid n \in \mathbb{N})$ ist die Konzeptklasse aller x-monotonen Manhattan-Kurvenzüge.

a) Gegeben seien zwei Punkte (a, b) und (u, v) aus $Bereich$, mit $a < u$ sowie eine Menge S von Punkten aus $Bereich$. Das Problem ist, einen x-monotonen Manhattan-Kurvenzug M mit der kleinsten Anzahl von Eckpunkten zu finden, so daß
   - M in (a, b) beginnt und in (u, v) endet,
   - stets in $Bereich$ verläuft und
   - keine Punkte aus S durchläuft.

   Konstruiere einen (in $|S|$ und n) polynomiellen Algorithmus, der das obige Problem löst, **vorausgesetzt** es existiert eine Lösung.

b) **Zeige:** C, mit obiger Namensfunktion, besitzt einen polynomiellen Occam-Algorithmus.

   Hinweis: Betrachte zuerst den Fall mit drei positven Punkten.

Aufgabe 7.2

Ein Manhatten-Kurvenzug M heißt nicht-überschneidend, wenn sich keine zwei Kanten von M überschneiden.

Setze $Bereich = \{0, 1, ..., 2^n - 1\} \times \{0, 1, ...2^n - 1\}$. Sei $\mathcal{D}_n$ die Konzeptklasse, die für jeden nicht-überschneidenden Manhattan-Kurvenzug M mit Eckpunkten aus $Bereich$ (und somit $M \subseteq Bereich$) ein Konzept d_M besitzt, nämlich die Menge aller Punkte (x, y) aus M.

Der Name von d_M ist wie in Aufgabe 7.1 gewählt.

Zeige: Die parametrisierte Konzeptklassse $\mathcal{D} = (\mathcal{D}_n \mid n \in \mathbb{N})$, mit obiger Namensfunktion, besitzt **keinen** effizienten PAC-Algorithmus, falls $R \neq NP$ gilt.

Hinweis: Reduziere das folgende NP-vollständige Problem auf das Konsistenzproblem von $\mathcal{D}$:
Gegeben sei ein Teilgraph des $n \times n$ Gitters (entstanden durch Entfernung von Gitterknoten und ihrer adjazenten Kanten). Gibt es einen Hamilton'schen Pfad durch alle übrigen Knoten?
(Das $n \times n$ Gitter besteht aus den n^2 Knoten (i, j), mit $1 \leq i, j \leq n$ und den $2n(n-1)$ Kanten $\{\{(i,j),(i+1,j)\} \mid 1 \leq i \leq n-1, 1 \leq j \leq n\} \cup \{\{(i,j),(i,j+1)\} \mid 1 \leq i \leq n, 1 \leq j \leq n-1\}$.
Ein Hamilton'scher Pfad in einem Graphen verbindet alle Knoten des Graphen, wobei jeder Knoten genau einmal besucht wird.)

Aufgabe 7.3

Seien $\mathcal{C}$ und $\mathcal{H}$ Konzeptklassen. Wir betrachten fehlertolerante Lernalgorithmen, die möglicherweise falsch klassifizierte Beispiele erhalten. In der Praxis tritt dieser Fall bei ungenauen Beobachtungen auf.

Für $\beta \in [0, 1]$ nehmen wir dabei an, daß ein Beispiel mit Wahrscheinlichkeit mindestens $1 - \beta$ richtig klassifiziert wird, und die Klassifikation des Beispiels mit Wahrscheinlickeit β unvoraussagbar ist.

Ein Lernalgorithmus L heißt genau dann β-fehlertolerant, wenn für alle ε ($0 < \varepsilon \leq 1$, mit $\beta \leq \frac{\varepsilon}{4}$), für alle δ ($0 < \delta \leq 1$), für alle $n \in \mathbb{N}$, für alle Verteilungen D auf $\Sigma^{\leq n}$ und für alle Konzepte $c \in \mathcal{C}$

$$prob[Fehler_D(c, h_c) \leq \varepsilon] \geq 1 - \delta$$

gilt, wobei $h_c \in \mathcal{H}$ die von L aufgestellte Hypothese ist. (Die Beispiele werden gemäß der unbekannten Verteilung D generiert. Für die Klassifikation der Beispiele benutzen wir das obige Fehlermodell mit Parameter β.)

Ein Algorithmus A heißt β-konsistent, wenn für alle ε (mit $\beta \leq \frac{\varepsilon}{4}$), δ, n, D und $c \in \mathcal{C}_n$

- A eine Anzahl s klassifizierten Beispielen anfordert und
- mit Wahrscheinlichkeit mindestens $1 - \frac{\delta}{2}$ eine Hypothese $h_c \in \mathcal{H}$ findet, die für höchstens $\frac{\varepsilon \cdot s}{2}$ der s angeforderten Beispiele nicht konsistent ist.

Bestimme eine untere Schranke s^* (als Funktion von ε, δ und $|\mathcal{H}|$), so daß ein β-konsistenter Algorithmus, der mindestens s^* Beispiele anfordert, ein β-fehlertoleranter

Lernalgorithmus ist.

Mit anderen Worten: „Fast-konsistente" Hypothesen garantieren die PAC-Eigenschaft selbst bei fehlerhaften Beispielen, solange die Fehlerrate β nicht zu groß ist ($\beta \leq \varepsilon/4$).

Hinweis: Sei h_c eine Hypothese mit $Fehler_D(c, h_c) > \varepsilon$. Zeige, daß
$prob[h$ stimmt mit einem zufällig gewählten Beipiel von c überein$] \leq \beta + (1-\beta)(1-\varepsilon)$.

Blatt 8

Aufgabe 8.1

Konstruiere für jedes β einen β-fehlertoleranten Lernalgorithmus für die Konzeptklasse Sym.
Hinweis: Nach Aufgabe 7.3 genügt es, einen β-konsistenten Algorithmus zu konstruieren (für $\beta \leq \frac{\varepsilon}{4}$). Versuche zu erreichen, daß die Anzahl der vom Algorithmus inkonsistent klassifizierten Beispielen (gemäß gegebener Klassifiziernung) die Anzahl der vom Fehlermodell falsch klassifizierten Beispielen nicht übersteigt.

Aufgabe 8.2

Wir definieren Schaltkreise mit Bausteinen vom Typ $\wedge$, $\vee$ mit Fan-in (Anzahl der Eingänge) 2 und vom Typ $\neg$ mit einem Eingang. Die Eingabe des Schaltkreises ist $x = (x_1, \ldots, x_n) \in \{0,1\}^n$. Schaltkreise lassen sich durch gerichtete azyklische Graphen beschreiben, wobei die Knoten des Graphen als $\wedge$, $\vee$ oder $\neg$ Baustein markiert sind. Eingang eines Bausteins kann jede Komponente der Eingabe x oder die Ausgabe eines „früheren" Bausteins sein.

Die Berechnung eines Schaltkreises verläuft wie folgt: Zuerst sind die Bausteine, deren Eingänge nur Eingabekomponenten sind, aktiv; alle anderen Bausteine sind passiv. Ein Baustein wird aktiv, sobald alle seine Vorgänger ihr Ergebnis berechnet haben. Ein aktiver Baustein berechnet dann seine Funktion auf den Ausgaben seiner Vorgänger. Das Ergebnis dieser einmaligen Berechnung steht dann allen Nachfolgern zur Verfügung. An den Senken (Bausteinen ohne Nachfolger) wird die Ausgabe produziert. Wir betrachten im folgenden nur Schaltkreise mit **einer** Senke und sagen, daß eine Eingabe akzeptiert wird, wenn die Senke des Schaltkreises den Wert 1 berechnet.

Die Tiefe $T(S)$ ist die Länge eines längsten Pfades von einer Eingabe zu einer Ausgabe. (Die Länge eines Pfades ist hierbei als die Anzahl seiner Kanten definiert.)

Eine Formel über den Variablen $x_1, \ldots, x_n$ ist wie folgt definiert:
- x_i und $\neg x_i$ $(1 \leq i \leq n)$ sind Formeln,
- wenn α und β Formeln sind, dann sind auch $(\alpha)\wedge(\beta)$, $(\alpha)\vee(\beta)$ und $\neg(\alpha)$ Formeln.

Die Länge $l\ddot{a}nge(\alpha)$ einer Formel α ist die Anzahl der logischen Operatoren $\wedge, \vee$ und $\neg$, die in α vorkommen.

a) **Zeige:** Für jeden Schaltkreis S der Tiefe t gibt es eine Formel α_S mit Formellänge höchstens $2^t - 1$, so daß S und α_S dieselben Eingaben akzeptieren.
Hinweis: Versuche eine Induktion über t.

b) Die Konzeptklasse $Formel_n$ besitzt für jede aussagenlogische Formel α, mit Variablenmenge $\{x_1, \ldots, x_n\}$ und $länge(\alpha) \leq n$, die Menge $wahr(\alpha)$ als Konzept. (Z.B. ist $wahr((x_1 \wedge \neg x_2) \vee (x_3 \wedge x_1)) \in Formel_4$.)

Zeige, daß die parametrisierte Konzeptklasse $Formel = (Formel_n \mid n \in \mathbb{N})$ keine polynomiellen PAC-Algorithmen besitzt, falls das RSA-System nicht gebrochen werden kann.

Aufgabe 8.3

Sei G ein gerichteter azyklischer Graph mit Fan-in 2 und genau einer Senke. Die Tiefe $Tiefe(G)$ von G bezeichnet die Länge eines längsten Pfades in G. (Die Länge eines Pfades ist hierbei als die Anzahl seiner Kanten definiert.)

Zeige: G kann mit höchstens $Tiefe(G) + 2$ Pebbles in Zeit höchstens $2^{Tiefe(G)+1} - 1$ gepebbelt werden.
Hinweis: Versuche eine Induktion über $Tiefe(G)$.

Aufgabe 8.4

Zeige: Für jeden (in Aufgabe 8.2 definierten) Schaltkreis G mit n Eingaben und Tiefe t (wobei $n \leq 2^t$) gibt es einen endlichen Automaten A_G mit höchstens $poly(2^t)$ Zuständen, so daß für alle $w \in \{0,1\}^n$:

$$A_G \text{ akzeptiert } \underbrace{w\, w\, \cdots w}_{2^{t+1}-1 \text{ mal}} \Leftrightarrow G \text{ akzeptiert } w.$$

Hinweis: Wende Aufgabe 8.3 an.

(Eine unmittelbare Konsequenz dieses Resultats ist, daß endliche Automaten keine polynomiellen PAC-Algorithmen besitzen, falls das RSA-System nicht gebrochen werden kann.)

Blatt 9

Aufgabe 9.1

Eine monotone DNF-Formel ist eine Disjunktion von monotonen Monomen. Die Konzeptklasse $Monoton\text{-}DNF_{t,n}$ besitzt für jede monotone DNF-Formel α mit höchstens t Termen über den Variablen $\{x_1, \ldots, x_n\}$ das Konzept $wahr(\alpha)$. Das leere Konzept wird durch die leere monotone DNF Formel dargestellt. Effiziente PAC-Algorithmen für $Monoton\text{-}DNF_{t,n}$ sind nicht bekannt. Wir betrachten deshalb ein stärkeres Lernmodell.

Sei $wahr(\alpha)$ das unbekannte Zielkonzept. Der Lernalgorithmus darf Elementfragen und Äquivalenzfragen stellen. In einer Elementfrage gibt der Lernalgorithmus ein Wort $x \in \{0,1\}^n$ aus und erhält $\alpha(x)$ als Antwort. In einer Äquivalenzfrage gibt der Lernalgorithmus eine monotone DNF-Formel β (seine gegenwärtige Hypothese) aus und erhält als Antwort entweder „die Hypothese ist richtig" oder ein Gegenbeispiel $x \in \{0,1\}^n$ (mit $\alpha(x) \neq \beta(x)$).

a) Ein monotones Monom m heißt Implikant von α, wenn $m(x) = 1 \Rightarrow \alpha(x) = 1$ für alle $x \in \{0,1\}^n$ gilt. Ein Implikant m von α heißt Primimplikant von α, wenn kein Teilmonom m' von m Implikant von α ist.

Zeige, daß $\alpha(x) = \bigvee_{m \text{ ist Primimplikant von } \alpha} m(x)$ für alle $x \in \{0,1\}^n$ gilt.

b) **Zeige**, daß jedes $\alpha \in Monoton\text{-}DNF_{t,n}$ höchstens t Primimplikanten besitzt.

c) **Konstruiere** einen Lernalgorithmus, der jedes Konzept aus $Monoton\text{-}DNF_{t,n}$ mit Hilfe von Elementfragen und Äquivalenzfragen in Zeit $poly(t,n)$ **exakt** bestimmt.

Aufgabe 9.2

Beweise, daß Konzepte aus der Konzeptklasse $Aut_{n,\{0,1\}}$ **nicht exakt** in polynomieller Zeit (bzgl. n) gelernt werden können, wenn **nur Elementfragen** gestellt werden dürfen. [In der Vorlesung werden wir zeigen, daß $Aut_{n,\{0,1\}}$ exakt in polynomieller Zeit gelernt werden kann, wenn Elementfragen **und** Äquivalenzfragen gestellt werden dürfen. Auch Äquivalenzfragen alleine sind nicht ausreichend für einen polynomiellen Lernalgorithmus, da sonst $Aut_{n,\{0,1\}}$ durch einen polynomiellen PAC-Algorithmus lernbar ist (siehe Aufgabe 5.3).]

Aufgabe 9.3

Die Konzeptklasse $Intervall_{t,n}$ besitzt für jede Vereinigung $I_1 \cup \cdots \cup I_s$ von höchstens t Intervallen über $\{0, 1, \ldots, 2^n - 1\}$ das Konzept $c_{I_1,\ldots,I_s} = I_1 \cup \cdots \cup I_s$ ($\subseteq \{0, 1, \ldots, 2^n - 1\}$).

Aus Aufgabe 6.1 können wir folgern, daß die Konzeptklasse $Intervall_{t,n}$ polynomielle PAC-Algorithmen besitzt.
Konstruiere einen Lernalgorithmus, der mit Hilfe von Element- und Äquivalenzfragen

jedes Konzept aus $Intervall_{t,n}$ in Zeit $poly(t,n)$ **exakt** bestimmt.
Hinweis: Binäre Suche ist hilfreich.

Aufgabe 9.4

Wir betrachten die Konzeptklasse $\mathcal{D}^*_{t,n}$. Für jeden (sich nicht überschneidenden) Manhattan-Kurvenzug $M \subseteq \{0, 1, \ldots 2^{n-1}\}^2$ mit höchstens t Eckpunkten besitzt $\mathcal{D}^*_{t,n}$ das Konzept d_M. Wir fordern allerdings zusätzlich, daß jeder Punkt von M mindestens zwei Nachbarpunkte besitzt, die nicht zu M gehören.

Konstruiere einen Lernalgorithmus, der mit Hilfe von Element- und Äquivalenzfragen jedes Konzept aus $\mathcal{D}^*_{t,n}$ in Zeit $poly(t,n)$ **exakt** bestimmt.
Hinweis: Die Vorgehensweise aus Aufgabe 9.3 ist hilfreich für die Konstruktion des Lernalgorithmus.

Blatt 10

Aufgabe 10.1

Entwirf einen Lernalgorithmus, der jedes Konzept aus k-CNF_n nur mit Hilfe von Äquivalenzfragen **exakt** bestimmt. Der Algorithmus soll in Zeit $O((2n)^{O(k)})$ laufen und höchstens $2^k \cdot \binom{n}{k}$ Äquivalenzfragen stellen.

Aufgabe 10.2 ??????????

In dieser Aufgabe betrachten wir Algorithmen, die Halbräume und konvexe Polygone (also Durchschnitte endlich vieler Halbräume) im $\mathbb{R}^2$ mit Hilfe von Element- und Äquivalenzfragen fehlerfrei erlernen.
Sei $Bereich = [0, 2^n - 1] \times [0, 2^n - 1] \cap \mathbb{Q} \times \mathbb{Q}$.

a) Wir betrachten die Konzeptklasse $Halbraum_2$, die für jede Hyperebene

$$H_{a,b,t} = \{(x, y) \mid ax + by = t\}$$

das Konzept

$$c_{a,b,t} = \{(x, y) \mid (x, y) \in Bereich,\ ax + by \geq t\}$$

enthält.

Konstruiere einen Lernalgorithmus, der für jedes Konzept $c_{a,b,t} \in Halbraum_2$ die Teilmenge $c_{a,b,t} \cap \mathbb{N}^2$ in Zeit $O(n)$ **exakt** bestimmt.

Es dürfen Elementfragen der Form: „Ist (x, y) ein Element des Konzepts?“ für $(x, y) \in Bereich$ gestellt werden. Gegenbeispiele auf Äquivalenzfragen sind stets Elemente aus $Bereich \cap \mathbb{N}^2$. (Falls $c_{a,b,t} \cap \mathbb{N}^2$ exakt bestimmt ist, wird somit kein Gegenbeispiel geliefert.)

b) Die Konzeptklasse *Konvexe-Polygone* besitzt für jedes konvexe Polygon P das Konzept

$$c_P = \{(x, y) \mid (x, y) \in Bereich,\ (x, y) \text{ ist innerer Punkt oder Randpunkt von } P\}.$$

Konstruiere einen Lernalgorithmus, der für jedes Konzept $c_P \in Konvexe\text{-}Polygone$ die Teilmenge $c_P \cap \mathbb{N}^2$ *exakt* bestimmt. Der Lernalgorithmus soll in Zeit $O(n \cdot k)$ laufen, wobei k die Anzahl der Kanten des zu lernenden Polygons ist.

Auch hier dürfen Elementfragen der Form: „Ist (x, y) ein Element des Polygons?“ für $(x, y) \in Bereich$ gestellt werden. Gegenbeispiele auf Äquivalenzfragen sind stets Elemente aus $Bereich \cap \mathbb{N}^2$.

Kommentar: Es kann gezeigt werden, daß konvexe Polytope (also Durchschnitte von endlich vielen Halbräumen) im $\mathbb{R}^d$ ebenfalls mit Hilfe von Element- und Äquivalenzfragen in polynomieller Zeit (in n, d und k) erlernbar sind. Ein entsprechendes Ergebnis für PAC-Algorithmen ist nicht bekannt.

Aufgabe 10.3

Es gibt einen Lernalgorithmus, der Konzepte aus der Konzeptklasse k-Term DNF_n mit höchstens $O(n \cdot 2^{O(k)})$ Elementfragen und Äquivalenzfragen **exakt** bestimmt. (PAC-Algorithmen benötigen $O((2n)^k)$ Beispiele.)

Wir konstruieren in dieser Aufgabe einen Lernalgorithmus, der ein Konzept $\alpha \in$ 2-Term DNF_n mittels $O(n)$ Element- und Äquivalenzfragen exakt bestimmt. Das zu lernende Zielkonzept $\alpha \equiv m_1 \vee m_2$ sei reduziert, d.h.:

- Keines der beiden Monome kann aus α entfernt werden, ohne daß $wahr(\alpha)$ sich ändert.
- Es gibt in keinem der beiden Monome ein Literal, das entfernt werden kann, ohne daß $wahr(\alpha)$ sich ändert.

a) Sei $x \in \{0,1\}^n$ ein positives Beispiel, das nur m_1 erfüllt. **Bestimme** ein Teilmonom m_1^* von m_1, das bis auf höchstens ein Literal alle Literale von m_1 besitzt. $O(n)$ Elementfragen und Äquivalenzfragen sind erlaubt.

b) **Beweise** die folgende Aussage: Falls $x = (x_1 \ldots x_n)$ ein negatives Beispiel für α ist, gibt es höchstens zwei Indizes i und j (mit $1 \leq i, j \leq n$, $i \neq j$), so daß $(x_1 \ldots x_{i-1}\ \bar{x}_i\ x_{i+1} \ldots x_n)$ und $(x_1 \ldots x_{j-1}\ \bar{x}_j\ x_{j+1} \ldots x_n)$ positive Eingaben für α sind.

c) Sei $x \in \{0,1\}^n$ ein positives Beispiel, das m_1 und m_2 erfüllt. **Konstuiere** einen Algorithmus, der ein positives Beispiel $y \in \{0,1\}^n$ findet, das genau ein Monom erfüllt. Der Algorithmus soll höchstens $O(n)$ Elementfragen und Äquivalenzfragen stellen.

d) **Zeige:** Sobald wir ein Monom von α kennen, können wir mit n Äquivalenzfragen das Zielkonzept α exakt erlernen.

e) **Konstuiere** einen Algorithmus, der jedes reduzierte Konzept aus 2-Term DNF_n mit höchstens $O(n)$ Elementfragen und Äquivalenzfragen **exakt** bestimmt.
Hinweis: Für den Fall, daß wir ein Monom aus α nur bis auf ein Literal bestimmen können, ist b) hilfreich. Wenn wir dann ein Monom aus α „fast“ kennen, ist die Idee aus d) nützlich.