Theorie des maschinellen Lernens

Hans U. Simon

24. Mai 2017

8 Effizientes PAC-Lernen

Wir erinnern daran, dass eine binäre PAC-lernbare Hypothesenklasse $\mathcal{H} = (\mathcal{H}_n)_{n\geq 1}$ *effizient* PAC-lernbar heißt, falls folgendes gilt:

1. $\mathrm{VCdim}(\mathcal{H}_n)$ ist polynomiell in n beschränkt (beherrschbare Informationskomplexität).
2. Es existiert ein PAC-Lernalgorithmus für $\mathcal{H}$, welcher, angesetzt auf eine Trainingsmenge $S \in Z_n^m$, eine Hypothese $h_S : \mathcal{X}_n \to \{0,1\}$ in Polynomialzeit berechnet (beherrschbare Berechnungskomplexität).

Im gesamten Kapitel machen wir die folgenden Voraussetzungen:

1. Die Informationskomplexität von $\mathcal{H}$ ist beherrschbar, d.h., $\mathrm{VCdim}(\mathcal{H}_n)$ ist polynomiell in n beschränkt.
2. Die Hypothesen $h \in \mathcal{H}_n$ besitzen eine „natürliche Kodierung“. Wir werden eine Hypothese $h \in \mathcal{H}_n$ oft (der Einfachheit halber) mit ihrem Codewort identifizieren. Die Länge dieses Codewortes wird dann als $|h|$ notiert. Wir setzen voraus, dass $\sup_{h\in\mathcal{H}_n} |h|$ polynomiell in n beschränkt ist.
3. Eine analoge Bemerkung gilt für die Beispielinstanzen aus $\mathcal{X}_n$.
4. Das Auswertungsproblem zu $\mathcal{H}$ — zu gegebenem $h \in \mathcal{H}_n$ und $x \in \mathcal{X}_n$ (mit beliebigem $n \geq 1$), berechne $h(x) \in \{0,1\}$ — ist in Polynomialzeit lösbar.

Wir werden in Abschnitt 8.1 an grundlegende Konzepte der Komplexitätstheorie erinnern. In Abschnitt 8.2 diskutieren wir zwei zu einer Klasse $\mathcal{H}$ assoziierte Probleme: das Konsistenzproblem und das „Minimum Disagreement Problem“. Es wird sich, grob gesprochen, ergeben dass $\mathcal{H}$ genau dann unter der Realisierbarkeitsannahme effizient PAC-lernbar ist, wenn das Konsistenzproblem zu $\mathcal{H}$ (evtl. randomisiert) in Polynomialzeit lösbar ist. Ein analoger Zusammenhang gilt für effiziente agnostische PAC-Lernbarkeit und das Minimum Disagreement Problem. In Abschnitt 8.3 stellen wir einige populäre Hypothesenklassen über dem Booleschen Grundbereich $\{0,1\}^n$ vor. Beispiele effizient PAC-lernbarer Klassen lernen wir in den Abschnitten 8.4 und 8.5 kennen. Inhärent harte Lernprobleme behandeln wir im Abschnitt 8.6. Dabei diskutieren wir sowohl über komplexitätstheoretische als auch über kryptographische Barrieren zum Lernen.

8.1 Ein Ausflug in die Komplexitätstheorie

Es bezeichne Σ ein Alphabet (= endliche Menge) und Σ^+ die Menge der nichtleeren über Σ gebildeten Strings. Wenn wir zum Beispiel von einem binären Alphabet $\Sigma = \{0, 1\}$ ausgehen, so ergibt sich $\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$. Wir definieren $\Sigma^* = \Sigma^+ \cup \{\lambda\}$ als die Erweiterung von Σ^+ um das sogenannte *leere Wort* λ (bestehend aus 0 Buchstaben).

Ein Algorithmus ist ein Rechenverfahren A, das, angesetzt auf einen Eingabestring $w \in \Sigma^*$, eine Ausgabe aus $A(w) \in \Sigma^*$ produziert. Bei einem Algorithmus für ein Entscheidungsproblem ist die Ausgabe entweder 0 (für die Antwort „Nein“) oder 1 (für die Antwort „Ja“). $L(A) = \{w \in \Sigma^* |\ A(w) = 1\}$ ist dann die *Sprache der von A akzeptierten Wörter*.

Wir setzen ein intuitives Verständnis für einen *elementaren Rechenschritt* in einem Algorithmus voraus. Eine formale Definition würde ein mathematisches Rechenmodell erfordern wie zum Beispiel die Turing-Maschine (TM) oder die Random Access Maschine (RAM). Solche Rechenmodelle werden in Vorlesungen wie „Theoretische Informatik“ oder „Datenstrukturen“ eingeführt und verwendet.

Um ein Gefühl für den Begriff eines elementaren Rechenschrittes zu bekommen, erläutern wir im Folgenden, grob und informell, was eine RAM in einem Rechenschritt tun kann:

- eine arithmetisch-logische Operation durchführen, d.h. zwei in Rechen- oder Indexregistern befindliche Zahlen miteinander verknüpfen (etwa mit den Operationen $+, -, *, /$ bzw. $\vee, \wedge, \neg$),
- einen Transportbefehl ausführen (etwa den Inhalt einer Speicherzelle in ein Register laden oder, umgekehrt, den Inhalt eines Registers in einer Speicherzelle ablegen),
- einen bedingten oder unbedingten Sprung zu einer anderen Kommandozeile des Rechenprogrammes durchführen,
- mit direkter oder indirekter Adressierung auf eine Speicherzelle zugreifen.

Speicherzellen können wir uns abstrakt wie eine Kollektion $x_1, x_2, \ldots$ von Variablen vorstellen. Die Variable x_i hat (per Definition) die Adresse i. Bei direkter Adressierung steht die Adresse i explizit im Programmtext. Bei indirekter Adressierung steht die Adresse i in einem Indexregister.[1]

Bei dem sogenannten *uniformen Kostenmaß* wird ein Rechenschritt mit Kosten 1 belegt. Bei dem *logarithmischen Kostenmaß* wird als Kostenterm die Bitlänge der beteiligten Zahlen veranschlagt (etwa Kosten k bei der arithmetisch-logischen Verknüpfung zweier k-Bit Zahlen). Das uniforme Kostenmaß könnte missbraucht werden, indem man riesige kombinatorische Strukturen (zum Beispiel die Adjazenzmatrix eines Graphen) in die Bits einer riesengroßen Zahl hineinkodiert. Das könnte dazu führen, dass ein einziger arithmetisch-logischer Rechenschritt einen großen Graphen durchmustert und modifiziert. In solchen Fällen ist das logarithmische Kostenmaß vorzuziehen. Solange aber nur Zahlen einer polynomiell (in der

[1] Da ein Programm aus einem endlichen Text besteht, könnte man ohne indirekte Adressierung nur konstant viele Speicherzellen nutzen. Der Algorithmus muss aber die Möglichkeit haben die Anzahl der verwendeten Speicherzellen von der Länge $|w|$ des Eingabestrings abhängig zu machen.

Eingabelänge n) beschränkten Bitlänge verwendet werden, ist das uniforme Kostenmaß für unsere Zwecke akkurat genug. Manchmal werden wir das uniforme Kostenmaß sogar bei arithmetischen Operationen auf den rellen Zahlen einsetzen (freilich ohne die unendliche Bitlänge mit unsauberen Tricks auszunutzen).

Die Eingabelänge: Formal betrachtet ist eine Eingabeinstanz für einen Algorithmus durch ein Eingabewort w über einem Grundalphabet Σ gegeben und $n = |w|$ ist dann die Eingabelänge. Wenn wir aber nur daran interessiert sind zu überprüfen, ob die Anzahl der zum Lösen des Problems erforderlichen Rechenschritte polynomiell in n beschränkt ist, können wir ebensogut „natürliche Maße“ für die Eingabelänge n verwenden, die sich von $|w|$ nur um ein Polynom unterscheiden. Zum Beispiel ist die natürliche Größe eines Graphen $G = (V, E)$ einfach die Anzahl $n = |V|$ seiner Knoten. Wenn das Codewort für den Graphen ein Bitstring wäre, der seiner Adjazenzmatrix A entspricht, dann wäre die Länge des Codewortes ungefähr n^2 (= Anzahl der Bits in der Matrix A). Da n und n^2 sich nur um ein Polynom (hier vom Grad 2) unterscheiden, wird es für unsere Zwecke keine Rolle spielen, ob wir die Größe der Eingabe mit n oder mit n^2 veranschlagen. Ein Markenzeichen eines polynomiell zeitbeschränkten Algorithmus ist es, dass er die Eingabegrößen (wie zum Beispiel die Knoten und die Kanten eines Graphen) nur beschränkt häufig inspiziert und bei jeder einzelnen Inspektion nur sehr elementare Aktionen ausführt.

Komplexitätsklassen. Als *formale Sprache* über dem Alphabet Σ bezeichnen wir jede Teilmenge von Σ^*. Zu einer formalen Sprache $L \subseteq \Sigma^*$ assoziieren wir folgendes Entscheidungsproblem, genannt das *Wort- oder Mitgliedschaftsproblem zur Sprache L*: gegeben $w \in \Sigma^*$, ist w ein Mitglied von L, d.h., gilt $w \in L$? Ein *Entscheidungsproblem* ist ein Problem, bei dem zu Eingabeinstanzen w nur die Antworten „Ja“ oder „Nein“ (bzw. 1 oder 0) möglich sind. Formale Sprachen und Entscheidungsprobleme sind zwei Seiten der selben Münze:

- Wir können $L \subseteq \Sigma^*$ mit dem Wortproblem zu L identifizieren.
- Wir können umgekehrt ein Entscheidungsproblem identifizieren mit der formalen Sprache bestehend aus allen Strings, welche Ja-Instanzen repräsentieren.

Eine *Komplexitätsklasse* enthält, salopp gesprochen, formale Sprachen bzw. Entscheidungsprobleme die „ungefähr“ den gleichen Aufwand an Rechenzeit (oder Speicherplatz) erfordern. In diesem Kapitel benötigen wir die Definition der Komplexitätsklassen P, NP und RP. P steht hierbei für „deterministisch Polynomialzeit“, NP für „nichtdeterministisch Polynomialzeit“ und RP für „Random P“:

die Klasse P: Wir sagen ein Algorithmus arbeitet in *Polynomialzeit*, wenn die Anzahl seiner Rechenschritte durch ein Polynom in der Eingabelänge nach oben beschränkt ist. Wir sagen eine Sprache $L \subseteq \Sigma^*$ gehört zur Klasse P genau dann, wenn ein Algorithmus existiert, der zu einem Eingabewort w in Polynomialzeit entscheidet, ob $w \in L$.

die Klasse NP: Wir sagen eine Sprache $L \subseteq \Sigma^*$ gehört zur Klasse NP genau dann, wenn ein Polynom q und eine Sprache $L' \in P$ existieren mit

$$L = \{w \in \Sigma^* | \ \exists u \in \Sigma^* : (|u| \leq q(|w|) \wedge (\langle u, w \rangle \in L'\} \ .$$

Hierbei steht $\langle u, w \rangle$ für eine (natürliche) Kodierung des Paares (u, w) in einen String über dem Alphabet Σ. Man bezeichnet u dann auch als ein *Zertifikat* für die Aussage $w \in L$. Die Bedingung $|u| \leq q(|w|)$ besagt, salopp ausgedrückt, dass das Zertifikat „kurz" zu sein hat. Die Bedingung $\langle u, w \rangle \in L'$ in Verbindung mit $L' \in P$ besagt, salopp ausgedrückt, dass ein Zertifikat für $w \in L$ „leicht überprüfbar" sein muss. Die Klasse NP besteht, so gesehen, aus allen Sprachen, deren Strings ein kurzes und leicht überprüfbares Zertifikat besitzen.

die Klasse RP: Die Definition der Klasse RP benutzt randomisierte Algorithmen, also Algorithmen, die Zugriff auf perfekte (und voneinander unabhängige) Zufallsbits haben. Dadurch wird bei einem randomisierten Algorithmus für ein Entscheidungsproblem die Ausgabe $A(w) \in \{0, 1\}$ zu einer $0, 1$-wertigen Zufallsvariable (also zu einer Bernoulli-Variable). Die Klasse RP besteht per Definition aus allen Sprachen L, für welche ein randomisierter Polynomialzeit-Algorithmus A existiert, so dass

$$\Pr[A(w) = 1] \begin{cases} = 0 & \text{falls } w \notin L \\ \geq \frac{1}{2} & \text{falls } w \in L \end{cases} \ . \qquad (1)$$

Mit anderen Worten: Strings außerhalb von L werden niemals akzeptiert und Strings innerhalb von L werden mit einer Wahrscheinlichkeit von mindestens $1/2$ akzeptiert. Ein Algorithmus A mit den genannten Eigenschaften heißt *Monte-Carlo Algorithmus* für L.

Die Bedingung, eine Eingabe $w \in L$ lediglich mit einer Wahrscheinlichkeit von mindestens $1/2$ zu akzeptieren, ist recht schwach. Sie kann aber verschärft werden wie folgt:

Bemerkung 8.1 *Für jede Konstante $k \geq 1$ gilt folgendes: ein Monte-Carlo Algorithmus A für L kann in einen Polynomialzeit-Algorithmus A_k transformiert werden, der die Bedingung*

$$\Pr[A(w) = 1] \begin{cases} = 0 & \textit{falls } w \notin L \\ \geq 1 - 2^{-k} & \textit{falls } w \in L \end{cases}$$

an Stelle der schwächeren Bedingung (1) erfüllt.

Beweis Algorithmus A_k setzt A k-mal auf den Eingabestring w an (mit jeweils „frischen" Zufallsbits) und akzeptiert genau dann, wenn mindestens eine der k Rechnungen von A auf Eingabe w akzeptierend ist. Es ist offfensichtlich, dass A_k die gewünschten Eigenschaften hat. **qed.**

Offensichtlich gilt

$$P \subseteq RP \subseteq NP \ .$$

Es wird vermutet, dass diese Inklusionen echt sind. „$P \neq NP$?" ist eine der berühmtesten offenen Fragen in der theoretischen Informatik.

Schwerste Probleme in NP. Die Theorie der schwersten Probleme in *NP* beruht auf folgendem Reduktionsbegriff:

Definition 8.2 (polynomielle Reduktion) *Für formale Sprachen L_1, L_2 über einem Alphabet Σ definieren wir: $L_1 \leq_{pol} L_2$ genau dann, wenn eine in Polynomialzeit berechenbare Abbildung $f : \Sigma^* \to \Sigma^*$ existiert mit*

$$\forall w \in \Sigma^* : w \in L_1 \Leftrightarrow f(w) \in L_2 \ .$$

Die Abbildung f wird dann die Reduktionsabbildung *für $L_1 \leq_{pol} L_2$ genannt.*

Bemerkung 8.3 *Die Relation $\leq_{pol}$ hat die folgenden Eigenschaften:*

1. *Sie ist transitiv, d.h., $L_1 \leq_{pol} L_2 \leq_{pol} L_3 \Rightarrow L_1 \leq_{pol} L_3$.*
2. *Aus $L_1 \leq_{pol} L_2$ und $L_2 \in P$ folgt, dass $L_1 \in P$.*
3. *Aus $L_1 \leq_{pol} L_2$ und $L_2 \in RP$ folgt, dass $L_1 \in RP$.*

Beweis

1. Wenn f eine Reduktionsabbildung für $L_1 \leq_{pol} L_2$ und g eine für $L_2 \leq_{pol} L_3$ ist, dann ist $g \circ f$ eine Reduktionsabbildung für $L_1 \leq_{pol} L_3$.
2. Das Problem „$w \in L_1$?“ kann in Polynomialzeit gelöst werden wie folgt:
   (a) Berechne $f(w)$.
   (b) Setze den Polynomialzeit-Algorithmus für L_2 auf $f(w)$ an und erhalte das Indikatorbit $b = \mathbb{1}_{[w \in L_2]}$.
   (c) Gib b als Indikatorbit für „$w \in L_1$“ aus.

   Wegen der Äquivalenzen $b = 1 \Leftrightarrow f(w) \in L_2 \Leftrightarrow w \in L_1$ ist der angegebene Algorithmus korrekt. Offensichtlich arbeitet er in Polynomialzeit.
3. Der Beweis der dritten Aussage ist analog zum eben geführten Beweis der zweiten Aussage.

qed.

Wir kommen nun zum zentralen Begriff der *NP*-Vollständigkeitstheorie:

Definition 8.4 (schwerste Probleme in NP) *Eine formale Sprache L_0 heißt NP-*hart*, falls jede Sprache $L \in NP$ polynomiell auf L_0 reduzierbar ist. Falls zusätzlich $L_0 \in NP$ gilt, dann heißt L_0 NP-*vollständig *oder auch eine* schwerste Sprache *in NP.*

Als unmittelbare Folgerung aus Bemerkung 8.3 und Definition 8.4 erhalten wir:

Bemerkung 8.5 *1. Falls L_1 NP-hart ist und $L_1 \leq_{pol} L_2$, dann ist auch L_2 NP-hart.*

2. Falls L_0 NP-hart ist und $L_0 \in P$, dann gilt $P = NP$.

3. Falls L_0 NP-hart ist und $L_0 \in RP$, dann gilt $RP = NP$.

Beispiel 8.6 (Vertex Cover (VC)) *Es ist bekannt, dass das folgende Problem ein schwerstes Problem in NP ist: gegeben ein Graph $G = (V, E)$ und eine Zahl $k \geq 1$, existiert in G ein „Vertex Cover" der Größe höchstens k, d.h, existiert eine Teilmenge I von V mit $|I| \leq k$, welche von jeder Kante aus E mindestens einen Randknoten enthält?*

Es ist klar, dass VC ein Mitglied der Klasse NP ist, da die „Ja-Instanzen" (G, k) ein kurzes leicht überprüfbares Zertifikat besitzen, nämlich das betreffende Vertex Cover I:

- Die natürliche Größe der Eingabe (G, k) ist die Anzahl $n = |V|$ der Knoten im Graphen. Ein Vertex Cover $I \subseteq V$ kann nicht mehr als n Knoten besitzen. Das Zertifikat I ist also kurz.
- Wir können die Menge E kantenweise durchmustern. Für jede Kante $\{i, j\}$ können wir leicht kontrollieren, dass I mindestens einen der Randknoten i, j enthält. Das Zertifikat I ist also leicht überprüfbar.

Warum ist das Problem VC also schwer? Nun die Schwierigkeit besteht darin, ein passendes Zertifikat effizient zu finden. Es gibt $\binom{n}{k}$ mögliche Teilmengen $I \subseteq V$ mit $|I| = k$. Für $k = n/2$ sind das

$$\binom{n}{n/2} \approx \frac{2^n}{\sqrt{n}}$$

(also exponentiell) viele. Ein Verfahren mit einer „exhaustive search" durch alle Kandidatenmengen würde nicht in Polynomialzeit arbeiten.

8.2 Grundlegende Resultate

Das *Konsistenzproblem* zu $\mathcal{H} = (\mathcal{H}_n)_{n \geq 1}$ ist definiert wie folgt:

als Entscheidungsproblem: Gegeben $S \in Z_n^m$, existiert eine Hypothese $h \in \mathcal{H}_n$ mit $L_S(h) = 0$?

als Konstruktionsproblem: Gegeben $S \in Z_n^m$, kläre, ob eine Hypothese $h \in \mathcal{H}_n$ mit $L_S(h) = 0$ existiert und, ggf., konstruiere eine solche.

Der Parameter n ist bei diesem Problem variabel. Eine Hypothese $h \in \mathcal{H}_n$ mit $L_S(h)$ ist eine zu S *konsistente* Hypothese, d.h., eine Hypothese, die auf S fehlerfrei ist. Beachte, dass ein Algorithmus für das Konstruktionsproblem eine ERM-Hypothese konstruiert, sofern eine zu S konsistente Hypothese in $\mathcal{H}_n$ existiert (also etwa für $S \sim \mathcal{D}_n^m$ unter der Realisierbarkeitsannahme). Die Entscheidungsproblem-Variante notieren wir im Folgenden als Kons-E($\mathcal{H}$). Die Konstruktionsproblem-Variante notieren wir als Kons-K($\mathcal{H}$).
Das *Minimum Disagreement Problem* zu $\mathcal{H} = (\mathcal{H}_n)_{n \geq 1}$ ist definiert wie folgt:

als Entscheidungsproblem: Gegeben $S \in Z_n^m$ und $k \geq 0$, existiert eine Hypothese $h \in \mathcal{H}_n$ mit $L_S(h) \leq k/m$? Mit anderen Worten: existiert eine Hypothese $h \in \mathcal{H}_n$, welche auf S maximal k Fehler macht?

als Konstruktionsproblem: Gegeben $S \in Z_n^m$ und $k \geq 0$, kläre, ob eine Hypothese $h \in \mathcal{H}_n$ mit $L_S(h) \leq k/m$ existiert und, ggf., konstruiere eine solche.

als Optimierungsproblem: Gegeben $S \in Z_n^m$, konstruiere eine Hypothese $h \in \mathcal{H}_n$ mit einer kleinstmöglichen empirischen Fehlerrate auf S.

Der Parameter n ist bei diesem Problem wieder variabel. Beachte, dass ein Algorithmus für die Optimierungsvariante eine ERM-Hypothese konstruiert. Die Entscheidungsproblem-Variante notieren wir im Folgenden als MinDis-E($\mathcal{H}$), die Konstruktionsproblem-Variante als MinDis-K($\mathcal{H}$) und die Optimierungsvariante als MinDis-Opt($\mathcal{H}$).

Den Zusammenhang zwischem diesen beiden kombinatorischen Problemen und effizientem PAC-Lernen klären die folgenden beiden Resultate:

Theorem 8.7 *1. Wenn Kons-K($\mathcal{H}$) in Polynomialzeit lösbar ist, dann ist $\mathcal{H}$ unter der Realisierbarkeitsannahme effizient PAC-lernbar.*

2. Wenn $\mathcal{H}$ unter der Realisierbarkeitsannahme effizient PAC-lernbar ist, dann gilt Kons-E($\mathcal{H}$) $\in RP$.

Beweis

1. Der Algorithmus zum Lösen des Problems Kons-K($\mathcal{H}$), angesetzt auf eine Trainingsmenge $S \sim \mathcal{D}_n^m$, konstruiert in Polynomialzeit (unter der Realisierbarkeitsannahme) eine Hypothese h mit $L_S(h) = 0$, also eine ERM-Hypothese. Da ERM, wie wir wissen, zum PAC-Lernen führt, folgt direkt, dass $\mathcal{H}$ unter der Realisierbarkeitsannahme effizient PAC-lernbar ist.

2. Es sei A ein unter der Realisierbarkeitsannahme effizienter PAC-Lernalgorithmus für $\mathcal{H}$. Wir wollen zeigen, dass sich aus A ein Monte-Carlo Algorithmus für das Problem Kons-E($\mathcal{H}$) destillieren lässt. Es sei $S \in Z_n^m$ die gegebene Menge von markierten Beispielen. Wir spezifizieren einen Algorithmus A' wie folgt:

   (a) Wir setzen $\varepsilon = 1/(m+1)$, $\delta = 1/2$ und wählen $\mathcal{D}_n$ als die uniforme Verteilung auf $S|_{\mathcal{X}}$.

   (b) Wir generieren mit Hilfe von S und $\mathcal{D}_n$ eine zufällige Trainingsmenge $T \sim \mathcal{D}_n^{m'}$ der für A erforderlichen Größe m'. Dabei setzen wir zur Generierung von $T|_{\mathcal{X}}$ Zufallsbits ein: jede Instanz in $T|_{\mathcal{X}}$ wird zufällig gemäß $\mathcal{D}_n$ aus $S|_{\mathcal{X}}$ ausgewählt. Die Labels in T werden einfach aus S übernommen.

   (c) Wir setzen den Lernalgorithmus A auf T an und erhalten eine Hypothese h_T.

(d) Wir testen[2], ob h_T auf S fehlerfrei ist. Falls dem so ist, akzeptieren wir die Eingabeinstanz S. Andernfalls verwerfen wir sie.

Die zentralen Beobachtungen sind wie folgt:

(a) Wenn in $\mathcal{H}$ keine zu S konsistente Hypothese existiert, dann wird S niemals (also für keine Wahl der eingesetzten Zufallsbits) akzeptiert.

(b) Wenn aber in $\mathcal{H}$ eine zu S konsistente Hypothese existiert, dann erfüllen $\mathcal{H}_n$ und $\mathcal{D}_n$ die Realisierbarkeitsannahme und es gilt folgendes:

i. Mit einer Wahrscheinlichkeit von mindestens $1 - \delta = 1/2$ gilt $L_{\mathcal{D}_n}(h_T) \leq \epsilon = 1/(m+1)$.

ii. Würde h_T auch nur einen einzigen Fehler auf S machen, dann würde (wegen unserer Wahl von $\mathcal{D}_n$) gelten: $L_{\mathcal{D}_n}(h) \geq 1/m > 1/(m+1)$.

iii. Insgesamt hat sich also ergeben, dass h_T mit einer Wahrscheinlichkeit von mindestens $1/2$ auf S fehlerfrei ist und daher wird S mit einer Wahrscheinlichkeit von mindestens $1/2$ akzeptiert.

Aus den Beobachtungen (a) und (b)iii. folgt leicht, dass A' ein Monte-Carlo Algorithmus für Kons-E($\mathcal{H}$) ist.

qed.

Theorem 8.8 *1. Wenn MinDis-Opt($\mathcal{H}$) in Polynomialzeit lösbar ist, dann ist $\mathcal{H}$ effizient agnostisch PAC-lernbar.*

2. Wenn $\mathcal{H}$ effizient agnostisch PAC-lernbar ist, dann gilt MinDis-E($\mathcal{H}$) $\in RP$.

Den Beweis zu Theorem 8.8, der völlig analog zum Beweis von Theorem 8.7 zu führen ist, lassen wir aus.

Folgerung 8.9 *1. Wenn Kons-E($\mathcal{H}$) NP-hart ist, dann ist $\mathcal{H}$ nicht einmal unter der Realisierbarkeitsannahme effizient PAC-lernbar (außer wenn $RP = NP$).*

2. Wenn MinDis-E($\mathcal{H}$) NP-hart ist, dann ist $\mathcal{H}$ nicht effizient agnostisch PAC-lernbar (außer wenn $RP = NP$).

Beweis Wir beschränken uns auf den Beweis der ersten Aussage, da der Beweis der zweiten Aussage völlig analog geführt werden kann. Es genügt zu zeigen, dass aus der *NP*-Härte von Kons-E($\mathcal{H}$) und der effizienten PAC-Lernbarkeit von $\mathcal{H}$ unter der Realisierbarkeitsannahme die Aussage $RP = NP$ gefolgert werden kann. Die effiziente PAC-Lernbarkeit von $\mathcal{H}$ unter der Realisierbarkeitsannahme impliziert gemäß Theorem 8.7, dass Kons-E($\mathcal{H}$) $\in RP$. Wenn aber ein einziges *NP*-hartes Problem, wie zum Beispiel Kons-E($\mathcal{H}$), zur Komplexitätsklasse

[2]Hier geht die Voraussetzung ein, dass das Auswertungsproblem zu $\mathcal{H}$ polynomiell lösbar ist.

RP gehört, dann ergibt sich mit der dritten Aussage in Bemerkung 8.5, dass $RP = NP$.

qed.

Die Resultate in diesem Abschnitt haben ergeben, dass sich die Frage der *effizienten* PAC-Lernbarkeit einer PAC-lernbaren binären Hypothesenklasse anhand der Komplexität zweier rein kombinatorischer Probleme entscheiden lässt, nämlich anhand des zu $\mathcal{H}$ assoziierten Konsistenz- und anhand des zu $\mathcal{H}$ assoziierten Mininimum Disagreement Problems.

8.3 Ein Zoo von Booleschen Hypothesenklassen

Wir definieren zunächst ein paar spezielle Boolesche Formeln, die mit den Operationen $\vee$ (für das logische Oder), $\wedge$ (für das logische Und) über n Booleschen Variablen $v_1, \ldots, v_n$ und ihren Negationen $\overline{v_1}, \ldots, \overline{v_n}$ gebildet werden:

Definition 8.10 *Es seien $v_1, \ldots, v_n$ Boolesche Variablen, die mit den Werten 0 (für FALSE) und 1 (für TRUE) belegt werden können. Ein* Literal *ist per Definition eine negierte oder unnegierte Boolesche Variable, d.h., ein Literal hat die Form $\overline{v_i}$ oder v_i. Eine* Klausel *ist eine „Veroderung“ von Literalen, also ein Ausdruck der Form $\ell_1 \vee \ldots \vee \ell_k$ mit $k \geq 0$, wobei $\ell_1, \ldots, \ell_k$ Literale sind. Ein* Term *ist eine „Verundung“ von Literalen, also ein Ausdruck der Form $\ell_1 \wedge \ldots \wedge \ell_k$ mit $k \geq 0$, wobei $\ell_1, \ldots, \ell_k$ Literale sind. Eine k-*Klausel *ist eine Klausel, in welcher maximal k Literale vorkommen. Eine* monotone Klausel *ist eine Klausel, in welcher ausschließlich unnegierte Variable vorkommen. Die Begriffe k-*Term *und* monotoner Term *sind analog zu verstehen. Eine* CNF-Formel *ist eine Verundung von Klauseln. „CNF“ steht dabei für „Conjunctive Normal Form“. Eine* DNF-Formel *ist eine Veroderung von Termen. „DNF“ steht dabei für „Disjunctive Normal Form“. Eine k-*CNF Formel *ist eine CNF-Formel in der ausschließlich k-Klauseln verundet werden. Eine k-*DNF Formel *ist eine DNF-Formel in der ausschließlich k-Terme verodert werden.*

Diese Typen von Booleschen Formeln führen zu den folgenden binären Hypothesenklassen:

Definition 8.11 *k-CNF$_n$ ist die Klasse aller Booleschen k-CNF Formeln über den Variablen $v_1, \ldots, v_n$. Weiter sei k-CNF $= (k\text{-CNF}_n)_{n\geq 1}$. k-DNF$_n$ ist die Klasse aller Booleschen k-DNF Formeln über den Variablen $v_1, \ldots, v_n$. Weiter sei k-DNF $= (k\text{-DNF}_n)_{n\geq 1}$. k-Clause-CNF$_n$ ist die Klasse aller CNF-Formeln, in denen maximal k viele Klauseln verundet werden. k-Term-DNF$_n$ ist die Klasse aller DNF-Formeln, in denen maximal k viele Terme verodert werden. Weiter sei k-Clause-CNF $= (k\text{-Clause-CNF}_n)_{n\geq 1}$ und k-Term-DNF $= (k\text{-Term-DNF}_n)_{n\geq 1}$.*

Die Klasse 1-CNF heißt auch auch die Klasse der *Booleschen Konjunktionen*. Sie enthält Formeln der Form $\ell_1 \wedge \ldots \wedge \ell_k$ für $k \geq 0$ viele Literale $\ell_1, \ldots, \ell_k$.

Jede Belegung der Booleschen Variablen $v_1, \ldots, v_n$ mit 0 oder 1 führt auf die offensichtliche Weise zu einem $0,1$-Wert einer über $v_1, \ldots, v_n$ gebildeten Formel. Dabei sind lediglich folgende Auswertungsregeln zu berücksichtigen:

1. Aus $v_i = 0$ folgt $\overline{v_i} = 1$ und aus $v_i = 1$ folgt $\overline{v_i} = 0$.

2. Eine Formel F der Form $F_1 \vee \ldots \vee F_k$ hat den Wert 1 genau dann, wenn mindestens eine der Formeln $F_1, \ldots, F_k$ den Wert 1 liefert. Dies impliziert $F = 0$ für den pathologischen Grenzfall $k = 0$.

3. Eine Formel F der Form $F_1 \wedge \ldots \wedge F_k$ hat den Wert 1 genau dann, wenn alle Formeln $F_1, \ldots, F_k$ den Wert 1 liefern. Dies impliziert $F = 1$ für den pathologischen Grenzfall $k = 0$.

Somit kann jede Boolesche Formel F über $v_1, \ldots, v_n$ als eine Boolesche Funktion $F : \{0,1\}^n \to \{0,1\}$ interpretiert werden und die in Definition 8.11 aufgeführten Formelklassen repräsentieren binäre Hypothesenklassen über dem Booleschen Grundbereich.

Definition 8.12 *Eine* Boolesche 1-Entscheidungsliste *über den Variablen $v_1, \ldots, v_n$ ist eine Liste der Form*

$$(\ell_1, b_1), \ldots, (\ell_k, b_k), (\ell_{k+1}, b_{k+1}) \ , \tag{2}$$

wobei $k \geq 0$, $b_1, \ldots, b_k, b_{k+1} \in \{0,1\}$, $\ell_1, \ldots, \ell_k$ sind Literale und $\ell_{k+1} = 1$. Die Paare (ℓ_i, b_i) heißen die Einträge *der Liste. Das spezielle Paar (ℓ_{k+1}, b_{k+1}) heißt der* Schlusseintrag. *Eine* 1*-Entscheidungsliste heißt* monoton*, wenn alle in ihr enthaltenen Literale unnegierte Boolesche Variable sind. Es bezeichne* 1*-DL_n die Klasse aller* 1*-Entscheidungslisten über $v_1, \ldots, v_n$ und* 1*-$DL = (1\text{-}DL_n)_{n \geq 1}$ bezeichne die zugehörige parametrisierte Hypothesenklasse.*

Wir sagen ein Vektor $a \in \{0,1\}^n$ wird von dem Paar (v_j, b) *absorbiert*, wenn $a_j = 1$. Er wird von dem Paar $(\overline{v_j}, b)$ absorbiert, wenn $a_j = 0$. Er wird also von einem Paar (ℓ, b) mit Literal ℓ absorbiert, wenn ℓ (interpretiert als Boolesche Formel) an der Stelle a zu 1 ausgewertet wird. Wir legen fest: ein Paar der Form $(1, b)$ mit $b \in \{0,1\}$ absorbiert jeden Vektor $a \in \{0,1\}^n$.

Eine Entscheidungsliste L der Form (2) wird als Boolesche Funktion $L : \{0,1\}^n \to \{0,1\}$ interpretiert wie folgt:

1. Zu gegebenem $a \in \{0,1\}^n$ bestimme den frühesten Eintrag in L, sagen wir den Eintrag (ℓ_i, b_i), in welchem a absorbiert wird. (Wegen $\ell_{k+1} = 1$ ist dies spätestens für $i = k+1$ der Fall.)

2. Setze dann $L(a) = b_i$.

Die Klasse der k-Entscheidungslisten, notiert als k-DL, ist analog zu 1-DL definiert mit dem Unterschied, dass die Einträge die Form (M, b) haben, wobei M ein k-Term ist (also ein Term, der maximal k Literale enthält).

8.4 Elementare effizient lösbare Lernprobleme

Eine Familie F von Teilmengen einer Grundmenge M ist durch die Teilmengenrelation $\subseteq$ partiell geordnet. Wenn F abgeschlossen unter der Durchschnittsbildung $\cap$ ist, dann existiert zu jeder Teilmenge $S \subseteq M$, die in mindestens einer Menge aus F enthalten ist, eine eindeutige kleinste S enthaltende Teilmenge, nämlich $\cap_{T \in F: S \subseteq T} T$.

Wie wir wissen können wir jede Teilmenge von M mit ihrer Indikatorfunktion identizieren, d.h., wir können eine Familie F solcher Mengen auch als eine Familie von auf M definierten $0,1$-wertigen Funktionen auffassen. Die Teilmengenrelation entspricht dann der folgenden Relation $\leq$:

$$f \leq g \Leftrightarrow (\forall x \in M : f(x) = 1 \Rightarrow g(x) = 1) \ .$$

Die Durchschnittsabgeschlossenheit entspricht nun der Abgeschlossenheit bezüglich Verundung $\wedge$.

Wir können diese Beobachtungen auf Hypothesenklassen $\mathcal{H}_n$ über einem Grundbereich $\mathcal{X}_n$ anwenden, sofern $\mathcal{H}_n$ bezüglich $\cap$ bzw. $\wedge$ abgeschlossen ist. Unter der Realisierbarkeitsannahme eröffnet dies die Möglichkeit, zu einer Trainingsmenge $S \in Z_n^m$ eine sehr spezielle ERM-Hypothese zu assoziieren:

- Es sei $S_+ = \{a |\ (a,1) \in S\}$ die Menge aller positiven Beispiele in S. Die Menge S_- aller negativen Beispiele aus S ist analog definiert.
- Wähle die kleinste Hypothese aus $\mathcal{H}_n$, im Folgenden notiert als $\langle S_+ \rangle$, die auf S_+ fehlerfrei ist.

Beachte, dass $\langle S_+ \rangle$ nicht nur auf S_+, sondern auch auf S_- (und somit auf ganz S) fehlerfrei sein muss:
Unter der Realisierbarkeitsannahme existiert eine zu S konsistente Hypothese $h \in \mathcal{H}$. Da diese insbesondere auf S_+ fehlerfrei sein muss, folgt wegen der Minimalität von $\langle S_+ \rangle$, dass $\langle S \rangle \leq h$. Da h Punkte aus S_- mit 0 klassifiziert, muss dies erst recht für $\langle S_+ \rangle$ gelten.
Algorithmen, die aus S die Hypothese $\langle S_+ \rangle$ berechnen, heißen „Closure Algorithmen“. Unsere kleine Diskussion lässt sich zusammenfassen wie folgt:

- Wenn alle Klassen $\mathcal{H}_n$ von $\mathcal{H} = (\mathcal{H}_n)_{n \geq 1}$ abgeschlossen unter $\cap$ bzw. $\wedge$ sind, und der Closure Algorithmus für $\mathcal{H}$ in Polynomialzeit implementierbar ist, dann ist das Problem Kons-K($\mathcal{H}$) in Polynomialzeit lösbar.

Dieses Argumentationsmuster wird uns gleich helfen, die erste Aussage des folgenden Resultates zu beweisen:

Theorem 8.13 *Das Problem Kons-K($\mathcal{H}$) ist für folgende Klassen $\mathcal{H}$ in Polynomialzeit lösbar:*

- *die Klasse der monotonen Booleschen Konjunktionen (also $\mathcal{H}$ = Monotone 1-CNF),*
- *die Klasse der monotonen 1-Entscheidungslisten (also $\mathcal{H}$ = Monotone 1-DL).*

Beweis Wir betrachten zunächst das Konsistenzproblem für monotone Boolesche Konjunktionen. Eine monotone Boolesche Konjunktion über den Variablen $v_1, \ldots, v_n$ hat die allgemeine Form $F_I = \wedge_{i \in I} v_i$. Wegen

$$\bigwedge_{i \in I} v_i \wedge \bigwedge_{j \in J} v_j = \bigwedge_{k \in I \cup J} v_k$$

sind monotone Boolesche Konjunktionen abgeschlossen unter $\wedge$. Wir müssen also lediglich zeigen, dass der Closure-Algorithmus für diese Klasse effizient implementiert werden kann. Beachte dazu, dass

$$F_I \leq F_J \Leftrightarrow J \subseteq I \ .$$

Die kleinste auf $\langle S_+ \rangle$ konsistente Hypothese aus Monotone 1-CNF ist daher gegeben durch die bezüglich $\subseteq$ größte Indexmenge $J \subseteq [n]$ einer auf $\langle S_+ \rangle$ fehlerfreien Hypothese F_J. Diese Menge kann wie folgt berechnet werden:

$$I := \{i \in [n] |\ \exists a \in S_+ : a_i = 0\}\ ;\ J := [n] \setminus I \ .$$

J entsteht, indem aus $[n]$ alle Indizes i entfernt werden, die zu einem Fehler auf $\langle S_+ \rangle$ führen würden. Da Indizes gleichsam nur unter Zugzwang entfernt werden, hat J die gewünschte Maximalitätseigenschaft.
Wir setzen den Beweis nun mit der Klasse der monotonen 1-Entscheidungslisten fort. Gegeben sei eine Menge $S \in Z_n^m$ markierter Beispiele. Es sei T eine Teilmenge von S. Wir sagen T ist *b-rein* mit $b \in \{0, 1\}$, wenn alle Beispiele in T das Label b haben. T heißt *rein*, wenn alle Beispiele in T das gleiche Label haben. Es bezeichne T_j die Menge aller markierten Beispiele aus T, deren Instanzen von Paaren der Form (j, b) absorbiert werden, d.h., $T_j = \{(a, b) \in T |\ a_j = 1\}$. Das folgende Verfahren bestimmt (wie wir noch zeigen werden) eine zu S konsistente 1-Entscheidungsliste, sofern eine solche überhaupt existiert:

1. Initialisiere L als leere Entscheidungsliste (ohne Einträge) und die Menge T als S.
2. Solange T nicht rein ist, durchlaufe folgende Schleife:
   (a) Finde, falls möglich, ein Paar (j, b) mit: T_j ist b-rein und $T_j \neq \emptyset$. Falls jedoch kein solches Paar existiert, dann gib die Meldung „Es gibt für S keine konsistente monotone 1-Entscheidungsliste“ aus und stoppe.
   (b) Erweitere L um den Eintrag (x_j, b) und setze $T := T \setminus T_j$.
3. Finde b mit T ist b-rein, erweitere L um den Eintrag $(1, b)$ und gib diese Liste dann aus.

Es ist offensichtlich, dass eine zu S konsistente Liste L ausgegeben wird, falls in Schritt 2(a) stets ein passendes Paar (j, b) gefunden wird. Es ist demnach noch Folgendes zu zeigen:

Behauptung: Falls eine zu S konsistente monotone 1-Entscheidungsliste L^* existiert, dann existiert bei jeder Ausführung von Schritt 2(a) ein passendes Paar (j, b).

Die Variable T des geschilderten Verfahrens enthält zu jedem Zeitpunkt die Beispiele aus S, deren Instanzen bisher nicht von L absorbiert werden. Es bezeichne L^+ die Liste L^* abzüglich ihres Schlusseintrages. Falls jeder Eintrag von L^+ auch in L vorkommen würde, dann würde L mindestens soviele Vektoren aus S absorbieren wie die Liste L^+. Dann müsste aber T zu diesem Zeitpunkt rein sein, so dass die Abbruchbedingung der Schleife erreicht und somit Schritt 2 verlassen würde. Schritt 2(a) würde dann nicht mehr erreicht. Wir dürfen daher annehmen, dass beim Erreichen von Schritt 2(a) die Liste L^+ einen Eintrag besitzt, der nicht bereits in L vorkommt. Es sei (x_j, b) der früheste solche Eintrag in L^+. Es sei T' die Menge der Beispiele aus S, deren Instanzen diesen Eintrag erreichen (also nicht vorher schon von L^+ absorbiert wurden). Dann ist T' natürlich b-rein. Da alle Einträge vor (x_j, b) in L^+ auch in L vorkommen, gilt $T \subseteq T'$. Daher ist T ebenfalls b-rein. Somit ist (j, b) ein passendes Paar, was den Beweis der Behauptung (und damit auch des Theorems) abschließt. **qed.**

Das Minimum Disagreement Problem ist leider schon für einfache Hypothesenklassen NP-hart, es sei denn man friert den Komplexitätsparameter n auf eine Konstante ein. Wir diskutieren im Folgendem ein konkretes Beispiel.

Definition 8.14 (achsenparallele Quader) *Es sei $\mathcal{B}_n$ die Klasse der achsenparallelen Quader, kurz n-*Boxen *genannt, über dem Grundbereich $\mathcal{X}_n = \mathbb{R}^n$. Weiter bezeichne $\mathcal{B}$ die zugehörige parameterisierte Klasse, d.h., $\mathcal{B} = (\mathcal{B}_n)_{n\geq 1}$.*

Beachte, dass die 2-Boxen gerade die achenparallelen Rechtecke in der euklidischen Ebene $\mathbb{R}^2$ sind.

Es ist nicht schwer zu zeigen, dass Kons-E($\mathcal{B}$) NP-hart ist. Hingegen ist sogar MinDIS-Opt($\mathcal{B}_n$) für eine *konstante Wahl des Komplexitätsparameters n* in Polynomialzeit lösbar. Wir illustrieren, das im Falle $n = 2$:

Lemma 8.15 *Das Problem MinDis-Opt($\mathcal{B}_2$) ist in Polynomialzeit lösbar.*

Beweis Gegeben sei eine Menge $S \in (\mathbb{R}^2 \times \{0, 1\})^m$ markierter Beispiele in der euklidischen Ebene. Es bezeichne S_+ die Menge der positiven (also mit 1 markierten) Beispielinstanzen in S. Es ist leicht zu argumentieren, dass eine 2-Box B, welche keinen Punkt aus S_+ enthält, kein Minimierer von $L_S(B)$ sein kann. Jede 2-Box B, welche mindestens einen Punkt aus S_+ enthält, können wir, ohne $L_S(B)$ zu vergrößern, so schrumpfen lassen, dass jede der vier Begrenzungskanten von B mindestens einen der Punkte aus S_+ enthält (kann im Extremfall 4-mal der selbe Punkt sein). Jede auf diese Weise normalisierte 2-Box ist auf die offensichtliche Art durch ein Quadrupel aus S_+^4 gegeben. Wir erhalten dann eine 2-Box mit kleinstmöglicher empirischer Fehlerrate auf S wie folgt:

1. Zu jedem Quadrupel $Q \in S_+^4$ bestimme die kleinste Q enthaltende 2-Box B_Q sowie ihre empirische Fehlerrate $L_S(B_Q)$.

2. Wähle ein Quadrupel Q^* aus, welches $L_S(B_Q)$ minimiert, und gib Q^* (als Repräsentation von B_{Q^*}) aus.

Die Laufzeit wird dominiert von der Anzahl aller Quadrupel aus S_+^4, maximal m^4 an der Zahl. **qed.**

Bemerkung 8.16 *1. Mit einem weniger naiven Algorithmus für das Problem MinDis-Opt($\mathcal{B}_2$) lässt sich (unter Verwendung dynamischer Programmierung) die Laufzeit verbessern von $O(m^4)$ auf $O(m^2 \log(m))$.*

2. Die Laufzeit eines naiven Algorithmus für das Problem MinDis-Opt($\mathcal{B}_n$) würde $O(m^{2n})$ betragen. Für ein konstant gehaltenes n ist dies immer noch ein Polynom in $m = |S|$.

8.5 Komplexere effizient lösbare Lernprobleme

Folgende Reduktionstechnik wird uns helfen, aus der effizienten Lernbarkeit einfacher Basisklassen, zuweilen die Lernbarkeit komplexerer Klassen zu folgern:

Definition 8.17 (polynomielle Lernreduktionen) *Es sei $\mathcal{H} = (\mathcal{H}_n)_{n\geq 1}$ bzw. $\mathcal{H}' = (\mathcal{H}'_n)_{n\geq 1}$ eine binäre Hypothesenklasse über dem Grundbereich $\mathcal{X} = (\mathcal{X}_n)_{n\geq 1}$ bzw. $\mathcal{X}' = (\mathcal{X}'_n)_{n\geq 1}$. Wir sagen $\mathcal{H}$ ist auf $\mathcal{H}'$* polynomiell L-reduzierbar*, notiert als $\mathcal{H} \leq^L_{pol} \mathcal{H}'$, wenn eine injektive und polynomiell beschränkte Abbildung $n \mapsto n'$ und für alle $n \geq 1$ Abbildungen $f_n : \mathcal{X}_n \to \mathcal{X}'_{n'}$, $g_n : \mathcal{H}_n \to \mathcal{H}'_{n'}$ und $g'_n : \mathcal{H}'_{n'} \to \mathcal{H}_n$ existieren, so dass folgendes gilt:*

1. *Für alle $n \geq 1$, alle $x \in \mathcal{X}_n$, alle $h \in \mathcal{H}_n$ sowie $x' = f_n(x)$ und $h' = g_n(h)$ gilt: $h(x) = h'(x')$.*
2. *Für alle $n \geq 1$, alle $x \in \mathcal{X}_n$, alle $h' \in \mathcal{H}'_{n'}$ sowie $x' = f_n(x)$ und $h = g'_n(h')$ gilt: $h'(x') = h(x)$.*
3. *Es seien $f : \cup_{n\geq 1}\mathcal{X}_n \to \cup_{n'\geq 1}\mathcal{X}'_{n'}$, $g : \cup_{n\geq 1}\mathcal{H}_n \to \cup_{n'\geq 1}\mathcal{H}'_{n'}$ und $g' : \cup_{n'\geq 1}\mathcal{H}'_{n'} \to \cup_{n\geq 1}\mathcal{H}_n$ die Abbildungen, deren Restriktionen auf $\mathcal{X}_n$ bzw. $\mathcal{H}_n$ oder $\mathcal{H}'_{n'}$ mit den Abbildungen f_n, g_n, g'_n übereinstimmen. Dann ist jede dieser drei Abbildungen in Polynomialzeit berechenbar.*

Die Abbildung f wird als Instanzentransformation *und die Abbildungen g und g' werden als* (erste und zweite) Hypothesentransformationen *bezeichnet.*

Die erste Definitionsbedingung sagt im Prinzip aus, dass eine simultane Anwendung von Instanzen- und erster Hypothesentransformation zur selben Markierung (mit 0 oder 1) führt. Die zweite Definitionsbedingung macht eine analoge Aussage für die zweite Hypothesentransformation an Stelle der ersten. Definition 8.17 ist motiviert durch die folgenden beiden Resultate:

Theorem 8.18 *Unter der Voraussetzung $\mathcal{H} \leq^L_{pol} \mathcal{H}'$ gilt folgendes:*

1. *Wenn Kons-E($\mathcal{H}$) NP-hart ist, so gilt dies auch für Kons-E($\mathcal{H}'$).*

2. *Wenn Kons-K($\mathcal{H}'$) in Polynomialzeit lösbar ist, so gilt dies auch für Kons-K($\mathcal{H}$).*

Beweis

1. Die Instanzentransformation f lässt sich wie folgt zu einer Beispielmengentransformation f^* fortsetzen:

$$S = [(x_1, b_1), \ldots, (x_m, b_m)] \mapsto S' = f^*(S) = [(x'_1, b_1), \ldots, (x'_m, b_m)] \ .$$

   Hierbei gilt $x'_i = f(x_i)$. Mit anderen Worten: wir wenden die Instanzentransformation f auf alle Beispielinstanzen x_i in S an und übernehmen alle Labels b_i aus S, ohne sie zu ändern. Es sei n die natürliche Zahl mit $S \in Z_n^m$. Die ersten beiden Bedingungen in Definition 8.17 bewirken, dass es zu jeder Hypothese $h \in \mathcal{H}_n$ eine Hypothese $h' \in \mathcal{H}'_{n'}$ mit $L_{S'}(h') = L_S(h)$ gibt, und umgekehrt. Insbesondere enthält $\mathcal{H}_n$ genau dann eine zu S konsistente Hypothese, wenn $\mathcal{H}'_{n'}$ eine zu S' konsistente Hypothese enthält. Die dritte Bedingung in Definition 8.17 bewirkt, dass sich die Abbildung $S \mapsto f^*(S) = S'$ in Polynomialzeit berechnen lässt. Damit ist f^* eine Reduktionsabbildung für Kons-E($\mathcal{H}$) $\leq_{pol}$ Kons-E($\mathcal{H}'$) und NP-Härte von Kons-E($\mathcal{H}$) vererbt sich auf Kons-E($\mathcal{H}'$).

2. Es sei A' ein Algorithmus, der in Polynomialzeit testet, ob zu einer gegebenen Menge $S' \in (Z'_n)^m$ eine konsistente Hypothese $h' \in \mathcal{H}'_n$ existiert und diese ggf. konstruiert. Wir transformieren A' in folgenden Algorithmus A:

   (a) Gegeben eine Menge $S \in Z_n^m$ berechne die Menge $S' = f^*(S) \in (Z'_{n'})^m$.

   (b) Setze A' auf S' an und erhalte entweder (Fall 1) eine Meldung „Es existiert keine zu S' konsistente Hypothese in $\mathcal{H}'_{n'}$“ oder (Fall 2) erhalte eine zu S' konsistente Hypothese $h' \in \mathcal{H}'_{n'}$.

   (c) Im Fall 1 mache eine entsprechende Meldung für S und $\mathcal{H}_n$. Im Fall 2 ermittle die Hypothese $h = g'_n(h')$.

   Beachte, dass h zu S konsistent ist, da h' zu S' konsistent ist. Es ist offensichtlich, dass der Algorithmus A das Problem Kons-K($\mathcal{H}$) in Polynomialzeit löst.

qed.

Den Beweis für das folgende Resultat, der völlig analog zum Beweis von Theorem 8.18 zu führen ist, lassen wir aus:

Theorem 8.19 *Unter der Voraussetzung $\mathcal{H} \leq_{pol}^{L} \mathcal{H}'$ gilt folgendes:*

1. *Wenn MinDis-E($\mathcal{H}$) NP-hart ist, so gilt dies auch für MinDis-E($\mathcal{H}'$).*

2. *Wenn MinDis-Opt($\mathcal{H}'$) in Polynomialzeit lösbar ist, so gilt dies auch für MinDis-Opt($\mathcal{H}$).*

Wir wenden unsere Reduktionstechnik jetzt auf zwei konkrete Hypothesenklassen an:

Theorem 8.20 *k-CNF $\leq^L_{pol}$ Monotone 1-CNF.*

Beweis Wir erinnern daran, dass k-CNF$_n$ die Klasse aller Verundungen von Klauseln mit maximal k Literalen bezeichnet. Die Klasse Monotone 1-CNF$_n$ enthält alle Verundungen von (unnegierten) Booleschen Variablen. In beiden Fällen werden die Formeln über n Booleschen Variablen $v_1, \ldots, v_n$ gebildet. Es sei $C_1, \ldots, C_{n'}$ eine (beliebige aber feste) Aufzählung aller Klauseln mit mindestens einem und höchstens k Literalen. Einfache Kombinatorik liefert

$$n' = \sum_{i=1}^{k} \binom{n}{i} 2^i = O(n^k) \ .$$

Offensichtlich ist die Abbildung $n \mapsto n'$ injektiv und polynomiell in n beschränkt. Es seien $v'_1, \ldots, v'_{n'}$ neue Boolesche Variable. Zu $a \in \{0,1\}^n$ und $C \in \{C_1, \ldots, C_{n'}\}$ bezeichne $C(a) \in \{0,1\}$ die Auswertung der Klausel C für die Belegung a der Booleschen Variablen $v_1, \ldots, v_n$. Dann verwenden wir die Abbildung

$$a \mapsto (C_1(a), \ldots, C_{n'}(a))$$

als Instanzentransformation. Als erste Hypothesentransformation verwenden wir die Abbildung

$$\wedge_{j=1}^{r} C_{i_j} \mapsto \wedge_{j=1}^{r} v'_{i_j} \ ,$$

d.h., wir ersetzen jede Klausel C_i, die in einer Formel aus k-CNF$_n$ vorkommt, durch die Boolesche Variable v'_i. Die umgekehrte Substitution, also Ersetzung von v'_i durch C_i liefert die zweite Hypothesentransformation. Man überzeugt sich leicht davon, dass die Bedingungen der Definition 8.17 erfüllt sind. **qed.**

Da wir bereits wissen, dass das Problem Kons-K($\mathcal{H}$) für die Klasse $\mathcal{H}$ der monotonen 1-CNF Formeln in Polynomialzeit lösbar ist, ergibt sich aus Theorem 8.18:

Folgerung 8.21 *1. Kons-K(k-CNF) ist in Polynomialzeit lösbar.*

2. k-CNF ist unter der Realisierbarkeitsannahme effizient PAC-lernbar.

Den Beweis für folgendes Resultat, der sehr ähnlich zum Beweis von Theorem 8.20 ist, lassen wir aus:

Theorem 8.22 *k-DL $\leq^L_{pol}$ Monotone 1-DL.*

Da wir bereits wissen, dass das Problem Kons-K($\mathcal{H}$) für die Klasse $\mathcal{H}$ der monotonen 1-Entscheidungslisten in Polynomialzeit lösbar ist, ergibt sich aus Theorem 8.18:

Folgerung 8.23 *1. Kons-K(k-DL) ist in Polynomialzeit lösbar.*

2. k-DL ist unter der Realisierbarkeitsannahme effizient PAC-lernbar.

8.6 Inhärent harte Lernprobleme

Die NP-Härte des Konsistenzproblems bzw. des Minimum Disagreement Problems zu einer binären Hypothesenklasse $\mathcal{H}$ ist, so wie wir es in Folgerung 8.9 präzisiert haben, eine Barriere auf dem Weg zum effizienten PAC-Lernen. Sie wird auch als die „komplexitätstheoretische Barriere“ bezeichnet. Diese Barriere lässt sich manchmal dadurch umgehen, dass wir dem Lerner erlauben, seine Hypothese aus einer Klasse $\mathcal{H}'$ zu wählen, welche ausdrucksstärker ist als die zu lernende Klasse $\mathcal{H}$. In Abschnitt 8.6.1 beschäftigen wir uns mit dieser Thematik. Es gibt jedoch eine gravierendere Barriere ohne diese Umgehungsmöglichkeit. Sie wird die „kryptographische Barriere“ genannt wird. Eine solche Barriere liegt vor, wenn wir zeigen können, dass die Abgabe einer Hypothese mit kleiner Fehlerrate dazu führt, dass ein als sicher geltendes Kryptosystem geknackt werden kann (zum Beispiel, indem wir eine Chiffre mit einem Bit des zugehörigen geheimen Klartextes labeln und diese Klassifikationslabels aus Beispielen lernen). Damit beschäftigen wir uns im Abschnitt 8.6.2.

8.6.1 Komplexitätstheoretische Barrieren

Wie früher bereits erwähnt sind Minimum Disagreement Probleme leider bereits für einfache Hypothesenklassen NP-hart. Wir illustrieren das anhand der Klasse Monotone 1-CNF:

Theorem 8.24 *MinDis-E(Monotone 1-CNF) ist NP-hart.*

Beweis Es sei VC das (als NP-hart bekannte) Vertex Cover Problem. Wir geben eine Reduktionsabbildung für VC $\leq_{pol}$MinDis-E(Monotone 1-CNF) an, woraus sich dann das Theorem unmittelbar erschließt. Eine Eingabeinstanz für VC besteht aus einem Graphen $G = (V, E)$ und einer Zahl $k \geq 1$. Wir setzen $n = |V|$ und setzen oBdA voraus, dass $V = [n]$, d.h., wir identifizieren die n Knoten von G mit den Nummern von 1 bis n. Die Frage lautet, ob eine Teilmenge $K \subseteq [n]$ mit $|K| \leq k$ existiert, die ein Vertex Cover ist, d.h., die von jeder Kante in E mindestens einen Randknoten enthält. Die Reduktionsabbildung soll daraus ein äquivalentes Minimum Disagreement Problem für die Klasse der monotonen 1-CNF Formeln machen. Für $K \subseteq [n]$ bezeichne $\vec{0}_K$ den Vektor aus $\{0,1\}^n$ der in den Positionen aus K Nulleinträge und außerhalb von K 1-Einträge hat. Statt $\vec{0}_{\{i\}}$ bzw. $\vec{0}_{\{i,j\}}$ schreiben wir aber einfach $\vec{0}_i$ bzw. $\vec{0}_{i,j}$. Wir bilden dann das Paar (G, k) auf folgende Eingabeinstanz (S, k) unseres Minimum Disagreement Problemes ab:

Knoten-Beispiele: Die Vektoren $\vec{0}_1, \ldots, \vec{0}_n$ erklären wir zu positiven Beispielen, d.h., sie erhalten das Label 1.

Kanten-Beispiele: Für jede Kante $\{i, j\} \in E$, erklären wir den Vektor $\vec{0}_{i,j}$ zu einem negativen Beispiel, d.h., er erhält das Label 0.

Die Menge S bestehe genau aus diesen Knoten- bzw. Kanten-Beispielen. Es genügt zu zeigen, dass folgendes gilt:

Behauptung: Es gibt genau dann ein Vertex Cover der Größe höchstens k in G, wenn es eine monotone 1-CNF Formel mit Booleschen Variablen $v_1, \ldots, v_n$ gibt, die auf S höchstens k Fehler macht.

Eine monotone 1-CNF Formel über den Variablen $v_1, \ldots, v_n$ hat die allgemeine Form $F_K = \wedge_{k \in K} v_k$ für eine Indexmenge $K \subseteq [n]$. Der Beweis der Behauptung basiert auf den folgenden zentralen Beobachtungen:

1. Der Vektor $\vec{0}_i$ wird von F_K genau dann korrekt gelabelt (also zu 1 ausgewertet), wenn $i \notin K$. Somit macht F_K auf den positiven Beispielen in S genau $|K|$ Fehler.

2. Ein Kantenbeispiel $\vec{0}_{i,j}$ wird von F_K genau dann korrekt gelabelt (also zu 0 ausgewertet), wenn $K \cap \{i, j\} \neq \emptyset$, d.h., wenn K mindestens einen Randknoten der Kante $\{i, j\}$ enthält.

Wir betrachten als erstes die Beweisrichtung „$\Rightarrow$“. Es sei also $K \subseteq [n]$ ein Vertex Cover in G. Aus unseren obigen zentralen Beobachtungen folgt sofort, dass F_K keine Fehler auf den Kantenbeispielen und $|K|$ Fehler auf den Knotenbeispielen macht, d.h., die Anzahl der Fehler von F_K ist nicht größer als die Anzahl $|K|$ der Knoten im Vertex Cover K.
Kommen wir schließlich zu der Beweisrichtung „$\Leftarrow$“. Es sei also eine Formel F_K mit $K \subseteq [n]$ gegeben, die auf S höchstens k Fehler macht. Falls ein Fehler auf einem Kanten-Beispiel $\vec{0}_{i,j}$ vorliegt, dann wird dieses von F_K fälschlicherweise mit 1 gelabelt. Dies ist nur dann möglich, wenn $i, j \notin K$. Wenn wir nun i in K aufnehmen, dann wird ein neuer Fehler auf $\vec{0}_i$ erzeugt (Label 0 statt Label 1), aber dafür wird $\vec{0}_{i,j}$ jetzt richtig mit 0 gelabelt. Die neue Formel hat also keine höhere empirische Fehlerrate auf S als die ursprüngliche. Diese kleine Überlegung zeigt, dass wir oBdA voraussetzen dürfen: erstens, F_K macht keine Fehler auf den Kanten-Beispielen und, zweitens, F_K macht höchstens k Fehler auf den Knoten-Beispielen. In Verbindung mit den obigen zentralen Beobachtungen können wir folgern: erstens, K ist ein Vertex Cover in G und, zweitens, $|K| \leq k$. Damit ist obige Behauptung, wie auch das Theorem, vollständig bewiesen. **qed.**

Für etwas komplexere Hypothesenklassen erweist sich auch das Konsistenzproblem zuweilen als NP-hart. Wir illustrieren das anhand der Klasse Monotone 2-Term DNF. Beachte, dass die Klasse Monotone 1-Term DNF übereinstimmt mit der Klasse Monotone 1-CNF: es handel sich beide Male um die Klasse der Verundungen von unnegierten Booleschen Literalen. Beim Übergang von Monotone 1-Term DNF zu Monotone 2-Term DNF geschieht also der Quantensprung von einem effizient lösbaren zu einem NP-harten Problem.

Theorem 8.25 *Kons-K(Monotone 2-Term DNF) ist NP-hart.*

Beweis Es sei 2-HG-Colorability (zu deutsch: 2-Färbbarkeitsproblem für Hypergraphen) das folgende (als NP-hart bekannte) Problem: gegeben $m, n \geq 1$ und $I_1, \ldots, I_m \subseteq [n]$, Können wir die Elemente $1, \ldots, n$ mit zwei Farben so einfärben, dass keine der Mengen $I_1, \ldots, I_m$ monochromatisch ist? Formal: existiert eine Abbildung $f : [n] \to \{1, 2\}$, so dass für alle

$i = 1, \ldots, m$ für die Bildmenge $f(I_i)$ gilt $f(I_i) = \{1, 2\}$?[3] Die Struktur $([n], \{I_1, \ldots, I_m\})$ wird auch als Hypergraph mit Knotenmenge $[n]$ und Hyperkanten $I_1, \ldots, I_m$ bezeichnet. Unsere Reduktionsabbildung soll die Frage der 2-Färbbarkeit dieses Hypergraphen in ein Konsistenzproblem für 2-Term DNF Formeln verwandeln. Wir geben eine passende Reduktionsabbildung an und verwenden dabei die im Beweis von Theorem 8.24 eingeführte Notation $\vec{0}_I$ mit $I \subseteq [n]$.

Knoten-Beispiele: Die Vektoren $\vec{0}_1, \ldots, \vec{0}_n$ erklären wir zu positiven Beispielen, d.h., sie erhalten das Label 1.

Hyperkanten-Beispiele: Für alle $i = 1, \ldots, m$ erklären wir den Vektor $\vec{0}_{I_i}$ zu einem negativen Beispiel, d.h., er erhält das Label 0.

Die Menge S bestehe genau aus diesen gelabelten Knoten- bzw. Hyperkanten-Beispielen. Es genügt zu zeigen, dass folgendes gilt:

Behauptung: Der Hypergraph $([n], \{I_1, \ldots, I_m\}$ ist genau dann 2-färbbar, wenn es eine zu S konsistente monotone 2-Term DNF Formel mit Booleschen Variablen $v_1, \ldots, v_n$ gibt.

Eine monotone 2-Term DNF über den Variablen $v_1, \ldots, v_n$ hat die allgemeine Form

$$F_{J,K} = \left(\bigwedge_{j \in J} v_j \right) \vee \left(\bigwedge_{k \in K} v_k \right)$$

für zwei Indexmengen $J, K \subseteq [n]$. Die zentralen Beobachtungen zum Beweis der Behauptung sind wie folgt:

1. Der Vektor $\vec{0}_i$ wird von $F_{J,K}$ korrekt gelabelt (also zu 1 ausgewertet), wenn $i \notin J \cap K$. Daher ist $F_{J,K}$ auf den Knoten-Beispielen genau dann fehlerfrei, wenn $J \cap K = \emptyset$.

2. Ein Vektor der Form $\vec{0}_I$ mit $I \subseteq [n]$ wird von $F_{J,K}$ genau dann zu 0 ausgewertet, wenn $I \cap J \neq \emptyset$ und $I \cap K \neq \emptyset$.

Wir betrachten nun die Beweisrichtung „⇒“. Sei also eine 2-Färbung von $1, \ldots, n$ gegeben, so dass keine der Mengen $I_1, \ldots, I_m$ monochromatisch ist. Sagen wir die Knoten aus $J \subseteq [n]$ sind „rot“ und die Knoten aus $K = [n] \setminus J$ sind „blau“ gefärbt. Da $J \cap K = \emptyset$, ist $F_{J,K}$ auf den Knoten-Beispielen fehlerfrei. Da keine der Mengen $I_1, \ldots, I_m$ monochromatisch ist, gilt $I_i \cap J \neq \emptyset$ und $I_i \cap K \neq \emptyset$ für alle $i = 1, \ldots, m$. Somit ist $F_{J,K}$ auch auf den Hyperkanten-Beispielen fehlerfrei.
Betrachten wir abschließend die Beweisrichtung „⇐“. Sei also eine zu S konsistente Formel $F_{J,K}$ mit $J, K \subseteq [n]$ vorgegeben. Da $F_{J,K}$ auf den Knoten-Beispielen fehlerfrei ist, folgt $J \cap K = \emptyset$. Da $F_{J,K}$ auch auf den Hyperkanten-Beispielen fehlerfrei ist, folgt $I_i \cap J \neq \emptyset$ und $I_i \cap K \neq \emptyset$ für alle $i = 1, \ldots, m$. Offensichtlich erhalten wir eine zulässige 2-Färbung,

[3] Wenn alle Mengen $I_1, \ldots, I_m$ aus exakt zwei Elementen bestünden, könnten wir sie als Kanten eines Graphen interpretieren, und wir erhielten das 2-Färbungsproblem für Graphen (welches effizient lösbar ist).

wenn wir alle Knoten aus J rot und alle Knoten aus $[n] \setminus J$ blau einfärben.[4] Damit ist obige Behauptung, wie auch das Theorem, vollständig bewiesen. **qed.**

Aus den Theoremen 8.24 und 8.25 können wir unmittelbar folgendes schließen:

Folgerung 8.26 *1. Die Klasse Monotone-1-CNF ist nicht effizient agnostisch PAC-lernbar (außer wenn $RP = NP$).*

2. Die Klasse Monotone 2-Term DNF ist nicht einmal unter der Realisierbarkeitsannahme effizient PAC-lernbar (außer wenn $RP = NP$).

Negativresultate dieser Art können manchmal umgangen werden, indem wir einem Algorithmus zum Lernen einer Klasse $\mathcal{H}$ erlauben, eine Hypothese zu wählen, die aus einer „mächtigeren" Klasse $\mathcal{H}'$ stammt. Dabei heißt $\mathcal{H}'$ *mächtiger* als $\mathcal{H}$, wenn es für jedes $n \geq 1$ und für jede Hypothese $h \in \mathcal{H}_n$ eine Hypothese $h' \in \mathcal{H}'_n$ gibt, welche die selbe binäre Funktion repräsentiert wie die Hypothese h. Die *effiziente PAC-Lernbarkeit von $\mathcal{H}$ mit Hypothesen aus $\mathcal{H}'$* ist dann auf die offensichtliche Weise definiert. Es gilt folgendes:

Lemma 8.27 *Wenn $\mathcal{H}'$ mächtiger ist als $\mathcal{H}$ und $\mathcal{H}'$ ist effizient agnostisch (bzw. effizient unter der Realisierbarkeitsannahme) PAC-lernbar, dann ist $\mathcal{H}$ effizient agnostisch (bzw. effzient unter der Realisierbarkeitsannahme) PAC-lernbar mit Hypothesen aus $\mathcal{H}'$.*

Beweis Wir führen den Beweis für PAC-Lernen unter der Realisierbarkeitsannahme. Da $\mathcal{H}'$ mächtiger ist als $\mathcal{H}$, ist eine Trainingsmenge $S \in Z_n^m$, deren Labels einer (dem Lerner unbekannten) Funktion $h \in \mathcal{H}_n$ entsprechen zugleich auch eine Trainingsmenge, deren Labels einer (dem Lerner unbekannten) Funktion $h' \in \mathcal{H}'_n$ entsprechen. Daher können wir einfach den PAC-Lernalgorithmus für $\mathcal{H}'$ auf S ansetzen.
Die Beweisführung für den Fall des effizienten agnostischen PAC-Lernens ist analog. **qed.**

Wir diskutieren ein Anwendungsbeispiel:

Lemma 8.28 *k-CNF ist mächtiger als k-Term DNF.*

Beweis Wir nutzen das für Boolesche Algebren gültige Distributivitätsgesetz aus:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \quad \text{und} \quad a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \ .$$

Diese Regel kann benutzt werden, um eine k-Term DNF Formel in eine äquivalente k-CNF Formel umzuformen. **qed.**

[4] Die Knoten aus $[n] \setminus (J \cup K)$ können natürlich beliebig eingefärbt werden, da bereits die Einfärbung der Knoten aus $J \cup K$ monochromatische Mengen I_i verhindert.

Beispiel 8.29 *Die 2-Term DNF Formel*

$$(v_1 \wedge \overline{v_3}) \vee (\overline{v_1} \wedge v_2 \wedge \overline{v_4})$$

repräsentiert die selbe Boolesche Funktion wie die 2-CNF Formel

$$(v_1 \vee \overline{v_1}) \wedge (v_1 \vee v_2) \wedge (v_1 \vee \overline{v_4}) \wedge (\overline{v_3} \vee \overline{v_1}) \wedge (\overline{v_3} \vee v_2) \wedge (\overline{v_3} \vee \overline{v_4}) \ .$$

Die Umformung beruht auf wiederholter Anwendung des Distibutivitätsgesetzes.

Da k-CNF unter der Realisierbarkeitsannahme effizient PAC-lernbar ist, erhalten wir unmittelbar folgendes Resultat:

Folgerung 8.30 *Die Klasse k-Term DNF ist unter der Realisierbarkeitsannahme effizient PAC-lernbar mit Hypothesen aus k-CNF.*

8.6.2 Kryptographische Barrieren

Wir starten mit der kleinen, aber feinen, Beobachtung, dass PAC-Lernalgorithmen (auch wenn sie Hypothesen außerhalb von $\mathcal{H}$ abliefern) verwendet werden können, um das unbekannte Label einer neuen Testinstanz (die im Training i.A. nicht gesehen wurde) mit kleiner erwarteter Fehlerrate vorherzusagen:

Lemma 8.31 *Wenn eine binäre Hypothesenklasse $\mathcal{H}$ unter der Realisierbarkeitsannahme effizient PAC-lernbar ist mit einer Klasse $\mathcal{H}'$ (deren Hypothesen in Polynomialzeit auswertbar sind), dann kann der zugrunde liegende Algorithmus (trainiert auf einer hinreichend großen Menge markierter Beispiele) benutzt werden, um in Polynomialzeit das Label zu einer bezüglich $\mathcal{D}$ zufälligen Testinstanz x mit einer Erfolgswahrscheinlichkeit von mindestens $1 - \alpha$ korrekt vorherzusagen. Dabei ist $\alpha > 0$ eine vorgegebene beliebig kleine positive Zahl.*

Beweis Es sei A der entsprechende Lernalgorithmus. Wir setzen $\epsilon = \delta = \alpha/2$, setzen A auf eine hinreichend große Trainingsmenge an und erhalten mit einer Wahrscheinlichkeit von mindestens $1 - \delta$ eine Hypothese h' mit einer Fehlerrate von maximal ε. Dann labeln wir die vorgegebene Testinstanz x mit $h'(x)$. Das korrekte Label sei b. Das Label $h'(x)$ ist korrekt, sofern nicht folgendes „schlechte Ereignis“ B eintritt:

$$L_{\mathcal{D}}(h') > \varepsilon \ \text{ oder } \ (L_{\mathcal{D}}(h') \leq \varepsilon \ \text{ und } \ h'(x) \neq b) \ .$$

Die Wahrscheinlichkeit von B ist beschränkt durch $\delta + \varepsilon = \alpha/2 + \alpha/2 = \alpha$. **qed.**

Ein Ausflug in die elementare Zahlentheorie. Es sei $N \geq 2$ eine natürliche Zahl und $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ seien die kleinsten Reste bei ganzzahliger Division durch N. Aus einer der Grundvorlesungen sollte bekannt sein, dass $(\mathbb{Z}_N, +, \cdot)$ ein Ring ist. Dabei wird Addition „+“ und Multiplikation „$\cdot$“ (sofern nicht ausdrücklich anders gesagt) immer modulo N ausgeführt. Weiterhin enthält

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N | \ \mathrm{ggT}(a, N) = 1\}$$

exakt die Menge der Elemente in $\mathbb{Z}_N$, welche ein multiplikatives Inverses besitzen. Mit Hilfe des erweiteren Euklidischen Algorithmus

- können wir effizient testen, ob $a \in \mathbb{Z}_N^*$
- und ggf. können wir effizient das Inverse b mit $a \cdot b = 1$ bestimmen.

$(\mathbb{Z}_N^*, \cdot)$ ist eine multiplikative Gruppe, die sogenannte Gruppe der primen Reste modulo N. Die Anzahl der Elemente in $\mathbb{Z}_N^*$ ist durch die Eulersche Funktion φ gegeben, d.h., $|\mathbb{Z}_N^*| = \varphi(N)$. Für alle $a \in \mathbb{Z}_N^*$ gilt $a^{\varphi(N)} = 1$. Hieraus folgt, dass man beim Rechnen in $(\mathbb{Z}_N^*, \cdot)$ in dem Exponenten einer Potenz modulo $\varphi(N)$ rechnen darf. Wir nehmen im Folgenden an, dass $\varphi(N)$ größer als 3, aber kein Vielfaches von 3, ist, so dass $\mathrm{ggT}(3, \varphi(N)) = 1$. Somit hat 3 modulo $\varphi(N)$ ein multiplikatives Inverses d mit $3 \cdot d = 1 \pmod{\varphi(N)}$. In diesem Fall ist die Abbildung $f_N : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ mit $f_N(x) = x^3$ eine Bijektion mit Umkehrabbildung $f_N^{-1}(x) = x^d$. Es gilt nämlich

$$f_N(f_N^{-1}(x)) = f_N^{-1}(f_N(x)) = x^{3d} = x \ ,$$

weil wir im Exponenten modulo $\varphi(N)$ rechnen dürfen und weil $3 \cdot d = 1 \pmod{\varphi(N)}$ gültig ist. Die Zahl $x = y^d \in \mathbb{Z}_N^*$ erfüllt demnach die Gleichung $x^3 = y$ und wird daher auch als die *diskrete Kubikwurzel* von y bezeichnet. Wenn N, y, d gegeben sind, dann ist $y^d \in \mathbb{Z}_N^*$ (mit der Methode des iterierten Quadrierens) effizient berechenbar. Es wird aber vermutet, dass die diskrete Kubikwurzel von y schwer zu berechnen ist, wenn lediglich N und y gegeben sind und wenn N das Produkt zweier großer Primzahlen ist:

Diskrete-Kubikwurzel-Vermutung (DKV) Es seien p, q zwei zufällig gewählte $n/2$-Bit Primzahlen, so dass für die n-Bit Zahl $N = pq$ gilt: 3 ist kein Teiler von $\varphi(N) = (p-1)(q-1)$. Es sei $y \in \mathbb{Z}_N^*$ zufällig gewählt. Die Vermutung lautet: es gibt keinen Algorithmus A mit einer in n polynomiell beschränkten Laufzeit, der die diskrete Kubikwurzel von y zu gegebenem (N, y) mit einer nicht vernachlässigbaren Erfolgswahrscheinlichkeit[5] korrekt berechnet. Die Wahrscheinlichkeit wird dabei genommen über die zufällige Wahl von p, q, y und (sofern A randomisiert ist) über die von A verwendeten Zufallsbits.

In dieser Vermutung bedeutet „zufällig“ stets „zufällig bezüglich der uniformen Verteilung“. Die DKV beruht auf der Vermutung, dass die Primfaktorzerlegungvon $N = pq$ ebenfalls schwer zu berechnen ist. Wenn nämlich die Teiler p, q von N bekannt wären, dann könnte die diskrete Kubikwurzel von y effizient berechnet werden wie folgt:

[5] d.h. mit einer Wahrscheinlichkeit von $1/p(n)$ für ein geeignetes Polynom p

1. Berechne $\varphi(N) = (p-1)(q-1)$.

2. Berechne das Inverse d von 3 modulo $\varphi(N)$ mit dem erweiterten Euklidischen Algorithmus.

3. Berechne $x = y^d$ mit der sogenannten Methode des iterierten Quadrierens.

Wir können im Rahmen dieser Vorlesung nicht auf den erweiterten Euklidischen Algorithmus und die Methode des iterierten Quadrierens eingehen. Es handelt sich um bekannte effiziente Methoden zur Bestimmung des multiplikativen Inversen und zum Potenzieren im Rahmen der modulo-Rechnung.

Berechnung diskreter Kubikwurzeln als Lernproblem Es sei $\mathcal{F}_n$ die Klasse aller Funktionen der Form

$$(N, y) \mapsto x \text{ mit } x^3 = y \text{ in } \mathbb{Z}_N^* \ ,$$

wobei gilt:

- $N = pq$ für zwei $n/2$-Bit Primzahlen p, q, so dass 3 kein Teiler von $\varphi(N) = (p-1)(q-1)$ ist.

- $y \in \mathbb{Z}_N^*$.

Wir stellen die Zahlen aus $\mathbb{Z}_N^*$ als n-Bit Zahlen dar. Für $x \in \mathbb{Z}_N^*$ bezeichne $x[i]$ das i-te Bit von x. Wir erhalten aus der Klasse $\mathcal{F}_n$ für jedes $i \in [n]$ eine binäre Hypothesenklasse $\mathcal{H}_{n,i}$, welche alle Funktionen der Form

$$(N, y) \mapsto x[i] \text{ mit } x^3 = y \text{ in } \mathbb{Z}_N^*$$

enthält. Weiter sei $\mathcal{H}^{DKV} = (\mathcal{H}_{n,i})_{n \geq i \geq 1}$.

Theorem 8.32 *Unter der DKV gilt folgendes: die Klasse $\mathcal{H}^{DKV}$ ist nicht effizient PAC-lernbar, und zwar selbst dann nicht, wenn die Realisierbarkeitsannahme gemacht wird und der Lerner Hypothesen aus einer von $\mathcal{H}^{DKV}$ verschiedenen Klasse abliefern darf.*

Beweis Es genügt zu zeigen, dass die DKV falsch ist, falls die Aussage dieses Theorems falsch ist (weil das im Umkehrschluss heißt, dass das Theorem unter der DKV korrekt ist). Angenommen wir verfügen über einen in Polynomialzeit arbeitenden Lernalgorithmus A. Wir verwenden diesen im Sinne von Lemma 8.31 als Hilfsmittel zum effizienten Voraussagen von Labels auf einer Testinstanz und können die diskrete Kubikwurzel x^* von $y^* \in \mathbb{Z}_N^*$ zu gegebenem N, y^* berechnen wie folgt:

1. Es sei n die Bitlänge von N. Wir setzen $\alpha := 1/n^2$.

2. Für jedes $i \in [n]$ präparieren wir eine hinreichend große Trainingsmenge S^i. Diese enthält Beipiele der Form $[(N, y), x[i]]$, wobei $x \in \mathbb{Z}_N^*$ zufällig gewählt und $y := x^3$ gesetzt wird (so dass x die diskrete Kubikwurzel von y ist).

3. Für jedes $i \in [n]$ setzen wir A auf S^i an und erhalten ein Voraussagebit $b[i]$ für $x^*[i]$.

4. Wir geben den Bitvektor $b = (b[1], \ldots, b[n])$ aus.

Wir beobachten zunächst, dass $y = x^3$ uniform auf $\mathbb{Z}_N^*$ verteilt ist, sofern x dies ist. Daher ist y im Trainingsbeispiel $[(N, y), x[i]]$ von Schritt 2 uniform zufällig aus $\mathbb{Z}_N^*$ (ebenso wie das Testbeispiel im Szenario der DKV)). Gemäß Lemma 8.31 ist für jedes $i \in [n]$ die Wahrscheinlichkeit für $b^*[i] \neq x^*[i]$ durch $\alpha = 1/n^2$ beschränkt. Gemäß der Union Bound ist dann die Wahrscheinlichkeit von $b \neq x^*$ durch $n\alpha = 1/n$ beschränkt. Somit wäre die Erfolgswahrscheinlichkeit sogar $1 - 1/n$ (also bei Weitem nicht vernachlässigbar) und die DKV wäre widerlegt. **qed.**

Kryptographisch harte natürliche Lernprobleme. Die Klasse $\mathcal{H}^{DKV}$ ist ein „Kunstprodukt“, das mit der Absicht entworfen wurde, ein hartes Lernproblem zu erzeugen. Wir wollen aber jetzt aufzeigen, dass sich das in Theorem 8.32 formulierte negative Ergebnis auf natürlicher gewählte Hypothesenklassen ausdehnen lässt. Zunächst beobachten wir, dass das Lernproblem zu $\mathcal{H}^{DKV}$ schwer bleibt, wenn wir es leicht modifizieren, indem wir die Beipielinstanzen nicht in der Form (N, y) präsentieren sondern in der redundanten Form

$$(N, z_0, z_1, \ldots, z_{n-1}) \text{ mit } z_i = y^{2^i} \text{ für } i = 0, 1, \ldots, n-1 \ . \tag{3}$$

Der Beweis von Theorem 8.32 muss dazu nur bei Schritt 2 des Kubikwurzelberechnungsverfahrens entsprechend modifiziert werden. Der Sinn dieses Manövers ist, dass sich die Kubikwurzel $x = y^d$ jetzt in einer sehr speziellen Form darstellen lässt: für $d = \sum_{i=0}^{n-1} d_i 2^i$ (also die Binärdarstellung von d) gilt

$$y^d = y^{\sum_{i:d_i=1} 2^i} = \prod_{i:d_i=1} y^{2^i} = \prod_{i:d_i=1} z_i \ . \tag{4}$$

Eine Funktion der Form

$$h_{N,d}(z) = \prod_{i:d_i=1} z_i$$

mit n-Bit Zahlen N, d sowie $z = (z_0, z_1, \ldots, z_{n-1})$ und $z_0, z_1, \ldots, z_{n-1} \in \mathbb{Z}_N^*$ wird als *iteriertes Produkt* in $\mathbb{Z}_N^*$ bezeichnet. Weiter sei $h_{N,d,i}$ die Funktion, die z auf das i-te Bit von $h_{N,d}(z)$ abbildet. Wir sagen eine Hypothesenklasse $\mathcal{H} = (\mathcal{H}_n)_{n \geq 1}$ *kann das iterierte Produkt darstellen*, wenn jede Funktion der Form $h_{N,d,i}$ in $\mathcal{H}_{n'}$ enthalten ist (bei geeigneter Wahl von n'). Eine solche Klasse kann nicht einfacher zu lernen sein als die Klasse $\mathcal{H}^{DKV}$ (mit der oben besprochenen modifizierten Darstellung der Beispielinstanzen). Damit ergibt sich unter der DKV folgendes:

Bemerkung 8.33 *Wenn eine Hypothesenklasse $\mathcal{H}$ das iterierte Produkt darstellen kann, dann ist $\mathcal{H}$ nicht effizient PAC-lernbar, und zwar selbst dann nicht, wenn die Realisierbarkeitsannahme gemacht wird und der Lerner Hypothesen aus einer von $\mathcal{H}$ verschiedenen Klasse abliefern darf.*

Aus der Komplexitätstheorie ist bekannt, dass die folgenden Klassen das iterierte Produkt darstellen können (ohne Beweis):

- Boolesche Schaltkreise (über der Standardbasis $\wedge, \vee, \neg$) einer polynomiell beschränkten Größe
- Boolesche Schaltkreise über der Basis der linearen Schwellenfunktionen (sogenannte Schwellen-Schaltkreise) einer polynomiell beschränkten Größe und einer konstanten Tiefe
- Deterministische Turing-Maschinen (DTM) mit einer logarithmischen Platzschranke (was, technisch gesehen, daraus folgt, dass Boolesche Schaltkreise von einer logarithmisch platzbeschränkten DTM auswertbar sind)

Schwache L-Reduktion und weitere Hiobsbotschaften. Eine *polynomielle schwache L-Reduktion* von einer Hypothesenklasse $\mathcal{H}$ auf eine Hypothesenklasse $\mathcal{H}'$ entsteht aus einer gewöhnlichen L-Reduktion durch Weglassen der zweiten Hypothesentransformation und durch Weglassen der Forderung, dass die erste Hypothesentransformation in Polynomialzeit berechenbar sein muss. Es ist nicht schwer, Folgendes zu zeigen:

Lemma 8.34 *Es sei $\mathcal{H}$ polynomiell schwach auf $\mathcal{H}'$ L-reduzierbar. Wenn $\mathcal{H}'$ mit Hilfe einer anderen Hypothesenklasse effizient (unter der Realisierbarkeitsannahme bzw. agnostisch) PAC-lernbar ist, dann trifft dies auch auf $\mathcal{H}$ zu.*

Im Umkehrschluss heißt das natürlich: Wenn $\mathcal{H}$ nicht effizient PAC-lernbar ist, dann trifft dies auch auf $\mathcal{H}'$ zu.

Lemma 8.35 *Die Klasse der logarithmisch platzbeschränkten DTM ist polynomiell schwach L-reduzierbar auf die Klasse der deterministischen endlichen Automaten (englisch: Deterministic Finite Automata oder kurz DFA) einer polynomiell beschränkten Größe.*

Beweis Wir skizzieren den Beweis lediglich. Wir nutzen aus, dass eine DTM M mit einer logarithmischen Platzschranke, angesetzt auf ein Eingabewort der Länge n, nur polynomiell in n viele verschiedene Konfigurationen haben kann. Die erste Hypothesentransformation bildet M ab auf einen DFA M', dessen Zustände 1–zu–1 den Konfigurationen von M für Eingabewörter w der Länge n entsprechen. Wir können aber nicht die triviale Instanzentransformation $w \mapsto w$ verwenden: der Kopf auf dem Eingabeband von M darf sich nämlich sowohl nach rechts als auch nach links bewegen, wohingegen der Kopf des DFA M' in jedem Rechenschritt eine Position nach rechts rücken muss. Wir lösen dieses Problem, indem wir die Eingabetransformation $w \mapsto w \dots w$ für eine ausreichende Anzahl von Duplikaten von w verwenden: wenn der Kopf auf dem Eingabeband von M eine Position nach links rückt, dann rückt der Kopf von M' um $n-1$ Positionen nach rechts. Auf diese Weise kann M' die Rechnung von M auf w simulieren. **qed.**

Wir ziehen ein deprimierendes Résumée:

Folgerung 8.36 *Die folgenden Klassen $\mathcal{H}$ sind nicht effizient PAC-lernbar, und zwar selbst dann nicht, wenn die Realisierbarkeitsannahme gemacht wird und wenn der Lerner Hypothesen aus einer von $\mathcal{H}$ verschiedenen Klasse abliefern darf:*

- *Boolesche Schaltkreise einer polynomiell beschränkten Größe*
- *Schwellen-Schaltkreise einer polynomiell beschränkten Größe und einer konstanten Tiefe*
- *logarithmisch platzbeschränkte deterministische Turing-Maschinen*
- *Deterministische endliche Automaten einer polynomiell beschränkten Größe*