Computational Intelligence

Prof. Dr. Wolfgang Maass
Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz

5. März 2008

Achtung: Dies Skriptum ist **nicht** geeignet für ein Selbststudium. Es fasst lediglich einige Definitionen und Formeln zusammen, welche in der Vorlesung erläutert werden. Das Lesen (oder auswendig lernen) dieses Skriptums ist nicht ausreichend für ein Verstehen der Materie (bzw. zum Bestehen der Prüfung). In der Prüfung wird auch Material aus der Vorlesung gefragt, welches nicht in diesem Skriptum steht.

1 Grundbegriffe des Maschinellen Lernens (ML)

Die Fähigkeit zum Lernen ist traditionell ein wichtiges Unterscheidungsmerkmal zwischen lebendigen Organismen und unbelebter Materie. In dieser LV untersuchen wir Methoden, die es dem Computer ermöglichen, diese Barriere zu durchstoßen und zu lernen. Um diese Methoden zu verstehen ist es zuerst einmal erforderlich, dass wir analysieren, was "Lernen" überhaupt ist. Wenn man genauer hinschaut, dann sehen wir, dass wir recht verschiedenartige Tätigkeiten als Lernen bezeichnen:

a) gesprochene Sprache verstehen lernen, oder lernen das Gesicht eines Freundes wiederzuerkennen, oder lernen aufgrund diagnostischer Messdaten eines Patienten zu beurteilen, welche Krankheiten dieser Patient hat, oder lernen eine Rezession vorherzusagen (→ **classification learning**).

b) lernen, einen Aktienkurs vorherzusagen (→ **regression**).

c) lernen, zu erkennen, wenn unser Auto ein ungewöhnliches Geräusch macht (→ **unsupervised learning**).

d) Surfing lernen, oder in Spiegelschrift schreiben lernen (→ **adaptive control, motor learning**).

e) lernen, einen vom Schilehrer gefahrenen Bogen in derselben Haltung nachzufahren (→ **imitation learning**).

f) eine erfolgreiche Strategie lernen, um ein Spiel zu gewinnen, oder in einem größeren Kontext: lernen zu überleben (→ **reinforcement learning**).

g) Vokabeln lernen (→ **memory management**).

Keine dieser Arten des Lernens sind unerreichbar für eine Maschine, aber sie erfordern jeweils verschiedene Techniken. In dieser LV werden wir uns auf a) und b) konzentrieren, also Lernen von Klassifikationen und Regression (beides sind Spezialfälle des sogenannten supervised learning) sowie c) (unsupervised learning).

Man könnte meinen, dass, wenn man einmal richtig verstanden hat, wie "Lernen" geht und dies dann im Computer implementiert hat, der Computer dann bald alle genannten Typen des Lernens beherrscht. Das ist bisher nicht gelungen. Ein Blick in die Biologie lässt die Schwierigkeit dieses Problems erahnen, weil es für jede Art von Lernen einige Tierarten gibt, die diese beherrschen und andere, die sie nicht beherrschen.

1.1 Ein Mathematisches Modell für Lernen (genauer gesagt: für Klassifikation und Regression)

Die Environment des Lerners wird modelliert durch ein Wahrscheinlichkeitsmaß (:= WM) P auf einer Menge der Form $A \times B$.[1] Jedes Paar $\langle a, b\rangle \in A \times B$ wird interpretiert als ein Beispiel eines konkreten Lernproblems. Im Fall eines Klassifikationsproblems ist B eine diskrete Menge. Zum Beispiel im Fall von Gesichtserkennung ist A die Menge aller Fotografien von Gesichtern, die vorkommen können und B ist die Menge der Namen von den Personen, deren Gesichter zu erkennen sind (möglicherweise ergänzt durch ein zusätzliches Element "don't know"). Ein Paar $\langle a, b\rangle \in A \times B$ ist in diesem Beispiel eine Fotografie a eines Gesichtes zusammen mit dem Namen der betreffenden Person. Das Wahrscheinlichkeitsmaß P gibt für jedes Paar $\langle a, b\rangle \in A \times B$ die Wahrscheinlichkeit $P(\langle a, b\rangle)$ an, mit der die Person b vor die betreffende Kamera tritt und das Bild a in der Kamera erzeugt.[2]

Lernen besteht in der einfachsten Version (**offline Lernen**) aus einer Trainingsphase und einer anschließenden Test- oder Performancephase. Während der Trainingsphase wird eine Liste $L = \langle\langle a_1, b_1\rangle, \langle a_2, b_2\rangle, \ldots, \langle a_l, b_l\rangle\rangle$ von l Trainingsbeispielen gemäß dem WM P aus $A \times B$ gezogen. Die Aufgabe des Lerners ist, aufgrund dieser Beispiele L eine Hypothese H_L zu erzeugen. Eine solche Hypothese ist mathematisch betrachtet eine Funktion von A nach B, also $H_L : A \to B$. Das ist sinnvoll, denn wenn in der anschließenden Testphase ein Bild a eines Gesichtes von der Kamera ausgegeben wird, so gibt $H_L(a) \in B$ einen (geratenen) Namen der Person an, um deren Gesicht es sich handelt.

[1] **Erinnerung**: $A \times B := \{\langle a, b\rangle : a \in A \text{ und } b \in B\}$ ist die Menge aller geordneten Paare mit der ersten Komponente aus A und der zweiten Komponente aus B.

[2] Wir nehmen in dieser LV meist an, dass jede für uns interessante Teilmenge M von $A \times B$ messbar ist, sodass deren Wahrscheinlichkeitsmaß $P(M)$ definiert ist. Oft kann man annehmen, dass A und B endliche Mengen sind, sodass ohne mathematische Komplikationen jede Teilmenge $M \subseteq A \times B$ als messbar betrachtet werden kann.

Die Qualität einer jeden Hypothese $H : A \to B$ wird für Klassifikationsprobleme anhand ihres sogenannten wahren Fehlers (true error) gemessen:

$$error_P(H) := P(\{\langle a, b\rangle : H(a) \neq b\}) \ .$$

In interessanten Anwendungen ist die Menge A in der Regel extrem groß, sodass man diesen **wahren Fehler** $error_P(H)$ einer Hypothese nicht genau bestimmen kann. Man kann aber eine Liste $T_k = \langle\langle a_i, b_i\rangle : 1 \leq i \leq k\rangle$ von k Testbeispielen gemäß P aus $A \times B$ ziehen und den **empirischen Fehler**

$$error_{T_k}(H) := \frac{\#\{i \in \{1, \ldots, k\} : H(a_i) \neq b_i\}}{k}$$

der Hypothese H für diese Liste T_k von k Testbeispielen ausrechnen. Es gilt

$$\lim_{k\to\infty} error_{T_k}(H) \ = \ error_P(H) \ ,$$

wobei aber die präzise Bedeutung dieser Aussage Begriffe aus der Wahrscheinlichkeitstheorie erfordert, weil die Terme $error_{T_k}(H)$ Zufallsvariablen bezeichnen.

Bei einem **Regressionsproblem** ist die Menge B in der Regel eine (recht große) Teilmenge von $\mathbb{R}$, z.B. $B = [0, 1]$. Es wird hier der "Fehler" einer Hypothese $H : A \to B$ in der Regel durch den mittleren quadratischen Fehler (MSE: mean squared error) gemessen. Dabei ist

$$MSE_P(H) := E((b - H(a))^2)$$

der **wahre mittlere quadratische Fehler** von H, und

$$MSE_{T_k}(H) := \frac{\sum_{i=1}^{k}(b_i - H(a_i))^2}{k}$$

der **empirische mittlere quadratische Fehler** einer Hypothese H auf einer Liste T_k von k Testbeispielen $\langle a_i, b_i\rangle$. Es gilt auch in diesem Fall, dass für jede Hypothese H $MSE_{T_k}(H)$ gegen $MSE_P(H)$ konvergiert für $k \to \infty$.

Gelegentlich betrachtet man auch simultane Regression für mehrere Variablen, z.B. mit $B = \mathbb{R}^k$ für $k > 1$. In diesem Fall ist eine Hypothese H eine Funktion von A nach $\mathbb{R}^k$. Wenn man b_j für die j-te Komponente eines Vektors $\mathbf{b} \in \mathbb{R}^{\mathbf{k}}$ schreibt, so kann man den wahren mittleren Fehler von H komponentenweise definieren:

$$MSE_P(H) := \sum_{j=1}^{k} E(b_j - H(a)_j)^2.$$

Diese Definition wird zum Beispiel benutzt um den Fehler eines Neuronalen Netzes mit mehreren Neuronen auf der letzten Schicht zu definieren.

Beachte: Erfolgreiches Lernen ist nur dann möglich, wenn das WM, das die Trainingsbeispiele erzeugt, das selbe ist (oder sehr ähnlich ist), wie das, welches die Testbeispiele erzeugt.

1.2 Analyse des wahren Fehlers $error_P(H)$ und des empirischen Fehlers $error_{T_k}(H)$ einer Hypothese H für ein Klassifikationsproblem mittels Methoden der Wahrscheinlichkeitstheorie

Fixiere eine beliebige Hypothese H. Dann kann man $p := error_P(H)$ auffassen als die Wahrscheinlichkeit, dass $H(a) \neq b$ gilt für ein zufällig (gemäß P) gezogenes Paar $\langle a, b\rangle$. Bei dem Vergleich von $H(a)$ und b handelt es sich um ein **Bernoulli-Experiment** ("Münzwurf") für eine Zufallsvariable Y, da nur die beiden Fälle $H(a) = b$ ($Y = 0$) und $H(a) \neq b$ ($Y = 1$) auftreten können. Der Erwartungswert $E(Y)$ hat den Wert $error_P(H) = p$, und die Varianz $Var(Y)$ den Wert $p \cdot (1 - p)$. Die Ermittlung des empirischen Fehlers $error_{T_k}(H)$ für eine Testmenge T_k, bestehend aus k gemäß P gezogenen Paaren $\langle a_i, b_i\rangle$, kann also aufgefasst werden als k unabhängige Wiederholungen eines Bernoulli-Experiments, d.h. man ermittelt den Wert von $X := \frac{Y_1+\ldots+Y_k}{k}$ für identisch (wie Y) verteilte unabhängige Zufallsvariablen $Y_1, \ldots, Y_k$. Dabei ist die Wahrscheinlichkeit, dass die Hypothese H bei k Testbeispielen genau m Fehler macht, gegeben durch die Binomialverteilung

$$\left(\begin{array}{c} k \\ m \end{array} \right) p^m \cdot (1-p)^{k-m} \ .$$

Dies ist die Wahrscheinlichkeit, dass $Y_1 + \ldots + Y_k = \sharp \{i \in \{1, \ldots, k\} : H(a_i) \neq b_i\}$ den Wert m hat. Ferner gilt $E(\sharp \{i \in \{1, \ldots, k\} : H(a_i) \neq b_i\}) = p \cdot k$ (weil der Erwartungswert stets additiv ist für Summen von Zufallsvariablen), also $E(error_{T_k}(H)) = E(X) = \frac{p \cdot k}{k} = p$, und
$Var(\sharp \{i \in \{1, \ldots, k\} : H(a_i) \neq b_i\}) = k \cdot p \cdot (1-p)$ (weil die Varianz additiv ist für Summen von **unabhängigen** Zufallsvariablen). Ferner ist $Var(X) = \frac{1}{k^2} Var(Y_1 + \ldots + Y_k)$, also $Var(X) = \frac{1}{k} \cdot p \cdot (1-p)$.

Gemäß der Tschebyschev-Ungleichung[3]

$$W(\mid X - \mu \mid \geq \varepsilon) \leq \frac{\sigma^2}{\varepsilon^2} \text{ für jede Zufallsvariable } X \text{ mit } E(X) = \mu \text{ und } Var(X) = \sigma^2$$

gilt für $X := error_{T_k}(H) \left(= \frac{Y_1+\ldots+Y_k}{k}\right)$ mit $E(X) = error_P(H)$ und $Var(X) = \frac{1}{k} \cdot error_P(H) \cdot (1 - error_P(H))$ dass

$$W(\mid error_{T_k}(H) - error_P(H) \mid \geq \varepsilon) \quad \leq \quad \frac{error_P(H) \cdot (1 - error_P(H))}{k \cdot \varepsilon^2} \quad .$$

Für jeden festen Wert von $\varepsilon > 0$ konvergiert diese Wahrscheinlichkeit gegen 0 für $k \to \infty$.

Mittels einer anderen Abschätzung (**Chernoff-Bound**) erhält man sogar einen exponentiell schnellen Abfall dieser Wahrscheinlichkeit mit wachsender Anzahl k von Beispielen in einer Testmenge T_k:

$$W(error_{T_k}(H) \geq error_P(H) + \varepsilon) \leq e^{-2k\varepsilon^2}$$

[3]folgt aus $W(\mid X - \mu \mid \geq \varepsilon) \cdot \varepsilon^2 \leq \int \mid X - \mu \mid^2 dW = \sigma^2$

und

$$W(error_{T_k}(H) \leq error_P(H) - \varepsilon) \leq e^{-2k\varepsilon^2}$$

also

$$W(|error_{T_k}(H) - error_P(H)| \geq \varepsilon) \leq 2\, e^{-2k\varepsilon^2}$$

für beliebige $k \in \mathbb{N}$ und $\varepsilon > 0$.

1.3 Anmerkungen

Ein Lerner (oder Lernalgorithmus) C ist in diesem mathematischen Modell für Lernen eine Funktion, die jeder Liste L von Beispielen aus $A \times B$ eine Hypothese $H_L : A \to B$ zuordnet.[4] Jeder praktische Lernalgorithmus kann nur Hypothesen $H_L : A \to B$ aus einer bestimmten Hypothesenklasse $\mathcal{H}$ erzeugen. Die Qualität eines Lernalgorithmus C wird zum einen daran gemessen, ob es in der Hypothesenklasse $\mathcal{H}$ aus der er Hypothesen erzeugt überhaupt eine Hypothese H gibt mit niedrigem wahren Fehler $error_P(H)$ ("expressibility of $\mathcal{H}$"). Zum anderen wird die Qualität eines Lernalgorithmus C daran gemessen, wie groß die Chance ist, dass C schon für eine relativ kurze Liste L von Trainingsbeispielen eine Hypothese H_L in $\mathcal{H}$ ausgibt, deren wahrer Fehler $error_P(H_L)$ nicht viel größer ist als der empirische Fehler $error_L(H_L)$. Jede einzelne dieser Anforderungen kann mit einer geeigneten Wahl der Hypothesenklasse $\mathcal{H}$ leicht erfüllt werden (wie?), aber es ist meist weniger leicht beide gemeinsam für eine Hypothesenklasse zu erfüllen.

Beachte, dass es für einen guten Lernalgorithmus C nicht ausreicht, dass $error_L(H_L)$ sehr klein ist, denn das ist oft relativ leicht zu erreichen, z.B. indem der Lerner die Beispiele in L "auswendig lernt", also

$$H_L(a) \;=\; \begin{cases} b, & \text{falls } \langle a, b\rangle \text{ in } L \\ \text{don't care}, & \text{sonst} \quad . \end{cases}$$

Ferner: Selbst wenn die Hypothesenklasse $\mathcal{H}$ beliebig groß ist, kann man nicht unbedingt $\inf_{H\in\mathcal{H}} error_P(H) = 0$ erwarten. Das WM P kann so geartet sein, dass es für manche $a \in A$ Werte $b_1, b_2 \in B$ mit $b_1 \neq b_2$ gibt sodass $P(\langle a, b_1\rangle) > 0$ und $P(\langle a, b_2\rangle) > 0$. Im medizinischen Bereich (Diagnose) tritt dies dann auf, wenn es zwei Personen gibt, die dieselben diagnostischen Messdaten haben, aber die eine hat eine Krankheit, die die andere Person nicht hat.

Validation Sets: Manchmal benutzt man beim maschinellen Lernen eine Teilmenge V der Trainingsbeispiele als validation set, d.h. als "interne Testmenge". Diese kann z.B. dazu benutzt werden um abzuschätzen, welche von zwei Hypothesen H_1, H_2 (die durch Anwendung von zwei verschiedenen Lernalgorithmen auf die Trainingsbeispiele in L, die nicht in V sind, erzeugt wurden) voraussichtlich auf zukünftigen (echten) Testdaten T weniger Fehler machen wird. Der Lerner C gibt dann diejenige der beiden Hypothesen

[4] In streng mathematischer Terminologie ist also ein Lernalgorithmus C eine Funktion von $(A\times B)^* := \bigcup_{l=1}^{\infty}(A \times B)^l$ nach B^A (B^A = Menge aller Funktionen von A nach B).

H_1, H_2 als die endgültige Hypothese H_L aus, welche auf V den geringeren Fehler hat. Validation sets werden z.B. bei der Crossvalidation (siehe Abschnitt 3.2) benutzt.

Typische Situation im Maschinellen Lernen: Es gibt zu wenige Trainingsbeispiele.

2 Lernen mit Neuronalen Netzen

Informationsverarbeitung im menschlichen Gehirn wird mittels ca. 10^{10} hierfür spezialisierten Zellen (Neurone oder Nervenzellen) durchgeführt. Das Input/Output Verhalten eines Neurons wird oft durch das folgende sehr einfache Modell simuliert (obwohl es die Arbeitsweise eines Neurons sehr ungenau wiedergibt, weil ein Neuron in Wirklichkeit Folgen von Pulsen, sogenannte spikes, ausgibt):

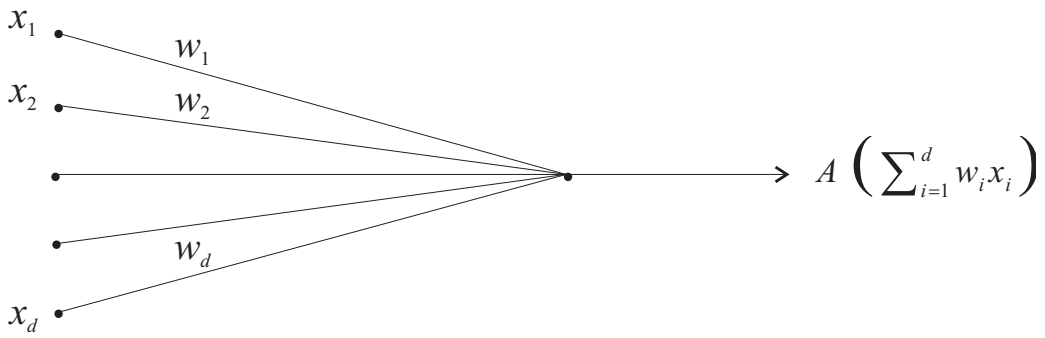


Abbildung 1: Input und Output eines künstlichen Neurons.

$w_1, \ldots, w_d$ sind interne Parameter (**Gewichte**) des Neurons, die beim "Lernen" verändert werden können und $A : \mathbb{R} \to \mathbb{R}$ ist eine vom Benutzer fest gewählte **Aktivierungsfunktion** des Neurons.
Inputs: d analoge (oder binäre) Zahlen $x_1, \ldots, x_d$
(Anmerkung: im Gehirn ist $d \approx 10^4$ für ein typisches Neuron).
Output: eine analoge (oder binäre) Zahl $A(\sum_{i=1}^{d} w_i x_i)$.

Als Aktivierungsfunktion A wählt man in der Regel eine der folgenden 3 Funktionen (siehe Abb. 2):

1. $A(x) = x$ (in MATLAB: purelin). Man bezeichnet das Neuron dann als **lineares Gatter**.
2. $A(x) = \frac{1}{1+e^{-x}}$ (in MATLAB: logsig). Man bezeichnet das Neuron dann als **sigmoides Gatter**.
3. $A(x) = \begin{cases} 1, & \text{falls } x \geq 0 \\ 0, & \text{sonst} \end{cases}$, (in MATLAB : hardlim).

Man bezeichnet diese dritte Funktion als **Sprungfunktion** (engl: Heaviside function) und das damit entstehende Gatter als **Schwellengatter** oder **McCulloch-Pitts Neuron** (nach McCulloch und Pitts, die dieses Neuronmodell 1943 als erstes Modell für die Informationsverarbeitungsfunktion eines biologischen Neurons vorschlugen). Beachte, dass

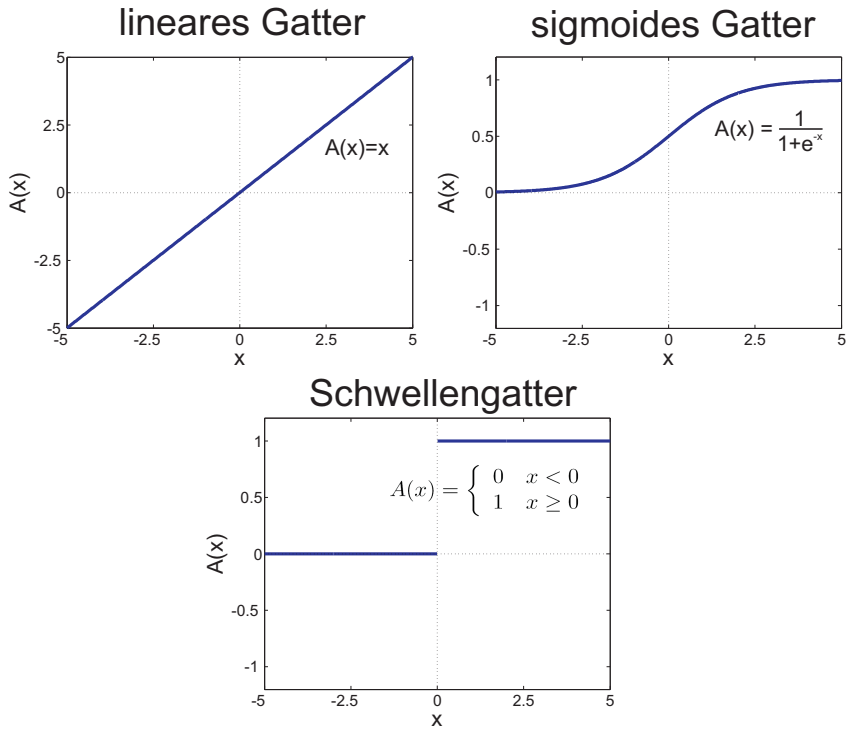


Abbildung 2: Aktivierungsfunktionen für verschiedene Gatter

diese Aktivierungsfunktion A als einzige binären Output liefert (selbst, wenn die Inputs $x_1, \ldots, x_d$ analog sind).

Schaltkreise, die aus Gattern einer dieser drei Arten (oder verwandten Arten) bestehen, werden als **künstliche Neuronale Netze** (kurz: NN) bezeichnet. Diese Schaltkreise haben mit den Schaltkreisen von Neuronen in unserem Gehirn gemeinsam, dass sie nicht programmiert werden. Stattdessen werden die Werte ihrer Gewichte – und damit ihr Input/Outputverhalten – durch einen Lernalgorithmus (angewendet auf eine Liste von Trainingsbeispielen) bestimmt.

In der Regel möchte man die Freiheit haben, einen Input-unabhängigen Offset w_0 zu definieren (dessen Wert auch durch den Lernalgorithmus bestimmt wird), sodass der Output des Neurons die Form $A(\sum_{i=1}^{d} w_i x_i + w_0)$ hat. Um die Schreibweise zu vereinfachen, nimmt man meist an, dass x_0 eine künstlich hinzugefügte 0-te Inputkomponente ist, die den konstanten Wert 1 hat (für **alle** Inputs). Die Defaultannahme ist daher im Folgenden, dass die erste Komponente a der Beispiele $\langle a, b\rangle$ die Form $a = \langle x_1, x_2, \ldots, x_d\rangle$ hat, dass das NN aber genau genommen den Input $\mathbf{x} = \langle 1, x_1, x_2, \ldots, x_d\rangle$ an Stelle von a erhält.[5] Unter

[5]Man kann zum Beispiel $\mathbf{x}^-$ für den ursprünglichen Vektor $\langle x_1, x_2, \ldots, x_d\rangle$ schreiben, wenn hierfür eine separate Notation erforderlich ist.

dieser Annahme ist

$$\sum_{i=1}^{d} w_i x_i + w_0 \;\; = \;\; \sum_{i=0}^{d} w_i x_i \;\; ,$$

sodass das Offset w_0 formal das Gewicht für die 0-te Inputkomponente x_0 darstellt. Da $\sum_{i=1}^{d} w_i x_i + w_0 \geq 0 \Longleftrightarrow \sum_{i=1}^{d} w_i x_i \geq -w_0$, wird $-w_0$ manchmal auch als **Schwelle** bezeichnet (siehe "Schwellengatter").

Ein **(künstliches) neuronales Netz** ist ein Schaltkreis bestehend aus künstlichen Neuronen. Es werden sowohl gerichtete (feedforward) Schaltkreise neuronaler Netze betrachtet, als auch rekurrente (recurrent) Schaltkreise mit Rückkopplungen. Wir untersuchen in dieser LV nur den erstgenannten einfacheren Typ, sogar nur einen Spezialfall, in dem der Schaltkreis aus endlich vielen Schichten $s = 1, \ldots, S$ besteht und alle Kanten von Neuronen auf einer Schicht s zu Neuronen auf Schicht $s+1$ verlaufen. Die Nummer S der letzten Schicht wird als **Tiefe** des neuronalen Netzes bezeichnet. Die Neuronen auf Schicht 1 erhalten die Input Variablen $x_1, \ldots, x_d$ als Input. Im Zeitschritt i wird der Output der Neurone in Schicht i berechnet. Der Output der Neurone auf der letzten Schicht S, der nach Zeitschritt S vorliegt, bildet den Output $\mathcal{N}(x_1, \ldots, x_d)$ eines neuronalen Netzes $\mathcal{N}$ für Input $\langle x_1, \ldots, x_d \rangle$. Aus historischen Gründen werden neuronale Netze mit $S > 1$ Schichten oft als **multi-layer-perceptrons** (MLP) bezeichnet.

Theorem 2.1 *Jede Boolesche Funktion $f : \{0,1\}^d \to \{0,1\}^m$ kann durch einen Schwellenschaltkreis (also durch ein neuronales Netz bestehend aus Schwellengattern) der Tiefe 2 berechnet werden.*

Theorem 2.2 *Zu jeder kontinuierlichen Funktion $f : [0,1]^d \to [0,1]^m$ und jedem $\varepsilon > 0$ gibt es ein neuronales Netz $\mathcal{N}$ der Tiefe 2, bestehend aus sigmoiden Gattern auf Schicht 1 und linearen (oder sigmoiden) Gattern auf Schicht 2, welches die Funktion f mit der Genauigkeit ε approximiert, d.h.*

für alle $\langle x_1, \ldots, x_d \rangle \in [0,1]^d$ gilt $\parallel f(x_1, \ldots, x_d) - \mathcal{N}(x_1, \ldots, x_d) \parallel \leq \varepsilon$.

2.1 Lösung von Klassifikationsproblemen mit neuronalen Netzen

Für ein binäres Klassifikationsproblem (d.h. P ist ein WM auf $A \times B$ mit $B = \{0,1\}$) kann man wie für Regressionsprobleme ein neuronales Netz mit einem Output Gatter trainieren. Man rundet den analogen Output eines linearen oder sigmoiden Output Gatters, um diesen als "Klassifikation" mit Werten in $\{0,1\}$ zu interpretieren. Für Klassifikationsprobleme mit $K > 2$ Klassen verwendet man meist ein neuronales Netz mit K Output Gattern und fasst jeweils die **Nummer** des Output Gatters, das für einen Netzwerk Input $\mathbf{x}$ den höchsten Wert ausgibt, als die Klassifikation von diesem Input $\mathbf{x}$ auf. Beim Trainieren wird der Soll-Output des betreffenden Gatters auf 1 gesetzt, die anderen auf 0.

Für binäre Klassifikationsprobleme kann man auch ein einzelnes Schwellengatter verwenden. Als Lernregel verwendet man dafür meist die **Perzeptron Lernregel:**

$$\mathbf{w}^{\text{neu}} = \mathbf{w}^{\text{alt}} - (A(\mathbf{w}^{\text{alt}} \cdot \mathbf{x}) - \mathbf{b}) \cdot \mathbf{x}$$

für Beispiele $\langle \mathbf{x}, b \rangle$ mit $b \in \{0, 1\}$, wobei A die Sprungfunktion ist.

Theorem 2.3 *Falls die Liste $L = \langle \langle a_1, b_1 \rangle, \ldots, \langle a_l, b_l \rangle \rangle$ von Beispielen aus $\mathbb{R}^d \times \{0, 1\}$ die Eigenschaft hat, dass die Mengen $M_1 := \{a_i : b_i = 1\}$ und $M_0 := \{a_i : b_i = 0\}$ linear trennbar sind, so macht die Perzeptron Lernregel beim wiederholten Durchlaufen der Liste L höchstens endlich viele Fehler. Sie findet also nach endlich vielen Schritten eine Hypothese H mit $error_L(H) = 0$.*

2.2 Regression mit einem linearen Gatter (= lineare Regression)

Sei nun P ein WM auf $A \times B = \mathbb{R}^d \times \mathbb{R}$. Die Beispiele sind dann von der Form $\langle a, b \rangle$ mit $a \in \mathbb{R}^d$ und $b \in \mathbb{R}$. Wir betrachten als erstes das Regressionsproblem mit der einfachsten, aber trotzdem sehr wichtigen Hypothesenklasse $\mathcal{H}$ bestehend aus der Menge aller **linearen Abbildungen** H von $\mathbb{R}^d$ nach $\mathbb{R}$. Das Ziel ist also, eine lineare Abbildung H zu finden, die $MSE_P(H)$ möglichst klein macht. Dabei kann jede lineare Abbildung $H : \mathbb{R}^d \to \mathbb{R}$ durch $d + 1$ Parameter $w_0, \ldots, w_d \in \mathbb{R}$ charakterisiert werden mit der Eigenschaft $H(x_1, \ldots, x_d) = \sum_{i=1}^{d} w_i x_i + w_0 = \sum_{i=0}^{d} w_i x_i$ (letztere Gleichung benutzt die vorher erläuterte Konvention mit $x_0 := 1$). Wir schreiben $H_{\mathbf{w}}$ für die so charakterisierte lineare Abbildung H.

Das Standardverfahren um eine Hypothese $H \in \mathcal{H}$ zu finden mit möglichst geringem Wert von $MSE_P(H)$ geht wie folgt vor:

1. Man zieht eine "genügend lange" Liste $L = \langle \langle \mathbf{x}(1), b_1 \rangle, \ldots, \langle \mathbf{x}(l), b_l \rangle \rangle$ von l Trainingsbeispielen[6] (wobei wir später verstehen lernen, was "genügend lang" heißt; als Daumenregel sollte l mindestens so gross sein wie die Anzahl der Parameter, durch die eine Hypothese H in $\mathcal{H}$ festgelegt wird – in diesem Fall also $d + 1$).

2. Man ermittelt eine Hypothese H in $\mathcal{H}$, für die $MSE_L(H)$ möglichst klein wird.

Den 2. Schritt kann man oft sehr direkt ausführen: indem man die Funktion $E(\mathbf{w}) := MSE_L(H_{\mathbf{w}})$ (für eine feste Liste L) mittels Gradientenabstieg minimiert. Man ersetzt also iterativ $\mathbf{w}^{\text{neu}} = \mathbf{w}^{\text{alt}} + \Delta\mathbf{w}$, mit $\Delta\mathbf{w} = -\alpha \, \nabla E(\mathbf{w})$ (wobei α die momentane Schrittweite des Gradientenabstiegs ist). Dazu ist erforderlich, dass der Gradient $\nabla E(\mathbf{w})$ existiert (und praktisch ausgerechnet werden kann). Das ist offensichtlich nur möglich, wenn $MSE_L(H_{\mathbf{w}})$ "glatt" von $\mathbf{w}$ abhängt, bei neuronalen Netzen mit linearen oder sigmoiden Gattern ist

[6] Die d Komponenten des i-ten Attributvektors $\mathbf{x}(i)$ werden mit $x_1(i), \ldots, x_d(i)$ bezeichnet.

dies aber stets der Fall. Im hier diskutierten konkreten Fall von Regression mit linearen Gattern gilt:

$$MSE_L(H_{\mathbf{w}}) = \frac{\sum_{i=1}^{l}(\mathbf{w} \cdot \mathbf{x}(i) - b_i)^2}{l} \ ,$$

also

$$\frac{\partial}{\partial w_j} MSE_L(H_{\mathbf{w}}) = \frac{2\sum_{i=1}^{l}(\mathbf{w} \cdot \mathbf{x}(i) - b_i) \cdot x_j(i)}{l} \ .$$

Um einen Schritt im Gradientenabstieg durchzuführen, kann man also einfach jedes Gewicht w_j^{alt} (für $j = 0, \ldots, d$) ersetzen durch

$$w_j^{\text{neu}} = w_j^{\text{alt}} - \alpha \cdot \sum_{i=1}^{l}(\mathbf{w}^{\text{alt}} \cdot \mathbf{x}(i) - b_i) \cdot x_j(i) \ ,$$

wobei $\alpha > 0$ die Schrittlänge des Gradientenabstiegs reguliert.[7] (**Frage:** Was sind mögliche Vor- und Nachteile einer großen Schrittlänge?).

Anstatt eines Batch-Updates kann man auch die Trainingsbeispiele einzeln durchgehen und **Online** für das i-te Trainingsbeispiel die Gewichtskorrektur

$$w_j^{\text{neu}} = w_j^{\text{alt}} - \alpha(\mathbf{w}^{\text{alt}} \cdot \mathbf{x}(i) - b_i) \cdot x_j(i), \qquad j = 0, \ldots, d$$

durchführen, für $i = 1, \ldots, l$. Dies bezeichnet man oft als **stochastischen Gradientenabstieg** für $MSE_L(H_{\mathbf{w}})$, weil die (zufällige) Reihenfolge der Trainingsbeispiele einen zusätzlichen Einfluss auf die Gewichtsänderung gewinnt.

[7] Der Faktor 2/l wurde hier in die Konstante α mit einbezogen.

Anmerkung:

1. Charakteristisch für die hergeleitete Lernregel ist, dass die Gewichtsänderung durch ein **Produkt** definiert wird, wobei der eine Faktor die **Abweichung des tatsächlichen Outputs des Gatters von dessen Soll-Output** beschreibt, der andere Faktor die **Input-Komponente** mit der dieses Gewicht multipliziert wird. Produkte dieser Art treten sehr häufig auf bei Lernregeln für neuronale Netze.

2. Für den Spezialfall von *linearen* Hypothesen $H_{\mathbf{w}}$ kann man in der Regel einen Vektor $\mathbf{w}$, der $MSE_L(H_{\mathbf{w}}) = (X\mathbf{w} - \mathbf{b})^T(X\mathbf{w} - \mathbf{b})$ minimiert, auch *explizit ausrechnen*: mittels der Formel
$$\mathbf{w} = (X^T \cdot X)^{-1} \cdot X^T \cdot \mathbf{b} \; .$$
Dies ist die Lösung der Gleichung $\nabla_{\mathbf{w}}(X \cdot \mathbf{w} - \mathbf{b})^T(X \cdot \mathbf{w} - \mathbf{b}) = \mathbf{0}$, weil $\nabla_{\mathbf{w}}(X\mathbf{w} - \mathbf{b})^T(X\mathbf{w} - \mathbf{b}) = 2X^T(X\mathbf{w} - \mathbf{b})$.
Hierbei ist X eine Matrix, deren Zeilen die Attributvektoren $\mathbf{x}(i)$ der Trainingsbeispiele $\langle \mathbf{x}(i), b_i \rangle$ sind, und
$$\mathbf{b} := \langle b_1, \ldots, b_l \rangle^T \; .$$
Es gilt: $X^T \cdot X$ ist invertierbar, falls die Spalten von X linear unabhängig sind.

3. Für *lineare* Hypothesen $H_{\mathbf{w}}$ gilt: Jeder Gewichtsvektor $\mathbf{w}$, der ein *lokales* Minimum von MSE$(H_{\mathbf{w}})$ bildet, ist gleichzeitig ein *globales* Minimum von $MSE(H_{\mathbf{w}})$.

2.3 Regression mit mehrschichtigen neuronalen Netzen: die Backprop Regel

Für viele in der Praxis auftretende WMe P auf $\mathbb{R}^d \times \mathbb{R}$ gibt es keine lineare Hypothese H die einen geringen Wert von $MSE_P(H)$ erzielt. Das Theorem 2.2 gibt in diesem Fall die Zuversicht dafür, dass mehrschichtige neuronale Netze Hypothesen H darstellen können, die wesentlich bessere Werte von $MSE_P(H)$ erzielen. Wenn alle Gewichte eines mehrschichtigen neuronalen Netzes $\mathcal{N}$ in einen Parametervektor $\mathbf{w}$ zusammengefasst werden, so kann man die durch das neuronale Netz $\mathcal{N}$ dargestellte Funktion (im Fall von d Inputs und 1 Output von $\mathcal{N}$) wieder als Funktion $H_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}$ schreiben. Lernen für ein solches neuronales Netz kann dann wieder zurückgeführt werden auf die Minimierung des empirischen Fehlers $E(\mathbf{w}) := MSE_L(H_{\mathbf{w}})$ für eine gemäß P gezogene Liste L von Trainingsbeispielen. Man führt wieder Gradientenabstieg aus, also $\mathbf{w}^{\text{neu}} = \mathbf{w}^{\text{alt}} - \alpha \cdot \nabla E(\mathbf{w})$. Der einzige Unterschied besteht darin, dass einige Komponenten $\frac{\partial}{\partial w_j} MSE_L(H_{\mathbf{w}})$ von $\nabla E(\mathbf{w})$ kompliziertere Formeln erfordern (gemäß der Kettenregel beim Differenzieren), weil der Netzwerk-Output $H_{\mathbf{w}}(x_1, \ldots, x_d)$ (= Output des Gatters auf der letzten Schicht S) nur indirekt von den Gewichten w_j von Gattern auf Schichten $s < S$ abhängt.

Bei mehrschichtigen neuronalen Netzen erreicht man beim Gradientenabstieg für $MSE(H_{\mathbf{w}})$ bestenfalls ein *lokales* Minimum (charakterisiert durch $\nabla_{\mathbf{w}} MSE(H_{\mathbf{w}}) = \mathbf{0}$). Es ist keine rechnerisch durchführbare Methode bekannt, mit der man stets zu einem *globalen* Minimum gelangt.

Backprop Lernregel für ein neuronales Netz mit S Schichten

Wir nehmen an, dass die Inputs eines jeden Neurons j auf einer Schicht s aus Outputs von Neuronen i auf Schicht $s-1$ (im Fall $s>1$) oder Input Variablen x_i (im Fall $s=1$) bestehen, wobei das betreffende Gewicht für diese Kante mit w_{ji} bezeichnet wird. Wir setzen $w_{ji} \equiv 0$ (also auch $\Delta w_{ji} = 0$) falls Neuron i (oder Input Variable x_i) keinen Input für Neuron j liefert.

Wir beschreiben hier zuerst die Online Version für ein einziges weiteres Trainingsbeispiel $\langle \langle x_1, \ldots, x_d \rangle, b \rangle$ aus $\mathbb{R}^d \times \mathbb{R}$:

1. Für jedes Neuron j setzen wir (für einen gewissen Parameter $\alpha > 0$):

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} + \alpha \cdot d(j) \cdot o(i) \ ,$$

   wobei $o(i)$ der Output des Neurons i für Netzwerk Input $\langle x_1, \ldots, x_d \rangle$ (bzw. $o(i) = x_i$, falls Neuron j auf Schicht 1 liegt) und $d(j)$ die "Fehlerzuweisung" für Neuron j darstellt.

2. $d(j) = (b - o(j)) \cdot A_j'(\sum_i w_{ji}^{\text{alt}} \cdot o(i))$ für das Output-Gatter j auf Schicht S.

3. $d(j) = \sum_k d(k) \cdot w_{kj}^{\text{alt}} \cdot A_j'(\sum_i w_{ji}^{\text{alt}} \cdot o(i))$ für alle anderen Gatter j.

Dabei ist A_j die Aktivierungsfunktion von Neuron j. In den Übungen wird gezeigt, dass diese Lernregel einen stochastischen Gradientenabstieg erzeugt, also $\alpha \cdot d(j) \cdot o(i)$ ist proportional zu $-\frac{\partial}{\partial w_{ji}}((H_{\mathbf{w}}(\mathbf{x}) - b)^2)$. Dabei ist die "Fehlerzuweisung" $d(j)$ für Neuron j proportional zu $-\frac{\partial}{net(j)}((H_{\mathbf{w}}(\mathbf{x}) - b)^2)$, wobei $net(j) := \sum_i w_{ji}^{\text{alt}} \cdot o(i)$ derjenige Ausdruck ist auf welchen die Aktivierungsfunktion A_j von Neuron j für den gegenwärtigen Netzwerk Input $\mathbf{x}$ angewendet wird. Durch "Nachdifferenzieren" erhält man den Faktor $o(i) = \frac{\partial}{\partial w_{ji}}(\text{net}(j))$, mit welchem $d(j)$ in der obigen Lernregel multipliziert wird.

Für die Offline Version von Backprop geht man durch eine Liste von Trainingsbeispielen und rechnet für jedes Trainingsbeispiel und jedes Gewicht w_{ji} die Gewichtsänderung $\Delta w_{ji} = w_{ji}^{\text{neu}} - w_{ji}^{\text{alt}}$ aus, implementiert sie aber noch nicht. Erst nachdem die ganze Liste durchgegangen wurde, führt man für jedes Gewicht w_{ji} die Summe aller vorher berechneten Δw_{ji} als Gewichtsänderung durch. Dieser gesamte Vorgang wird als **Epoche** (epoch) beim Lernen bezeichnet. In der Regel muss man recht viele Epochen durchführen, bis ein festgelegtes **Abbruch-Kriterium** (stopping criterion) erfüllt ist. Ein häufig verwendetes Abbruch-Kriterium benutzt eine zusätzliche gemäß dem WM P gezogene Liste V von Trainingsbeispielen **(validation set)**. Das Trainieren wird abgebrochen, sobald der empirische MSE auf V signifikant zu steigen beginnt (diese Regel wird oft als **early stopping** bezeichnet).

Frage: Wie könnte man das Steigen des MSE auf V rational erklären?

Speedup von Backprop

In der Regel ersetzt man die Lernregel $w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} + \alpha \cdot d(j) \cdot o(i)$ durch

$$w_{ji}^{\text{neu}} = w_{ji}^{\text{alt}} + \alpha \cdot d(j) \cdot o(i) + \beta \cdot (w_{ji}^{\text{alt}} - w_{ji}^{\text{uralt}}) \ ,$$

wobei w_{ji}^{uralt} der Wert von w_{ji} war, bevor er zu w_{ji}^{alt} geändert wurde ($\beta > 0$ ist eine geeignete Konstante). Durch diesen zusätzlichen **Impuls-Term (momentum term)** werden gemeinsame Trends in mehreren aufeinanderfolgenden Gewichtsänderungen verstärkt und sich widersprechende abgeschwächt. Visualisierung des Gradientenabstiegs mittels Höhenlinien für die durch $E(\mathbf{w})$ definierte Hügellandschaft:

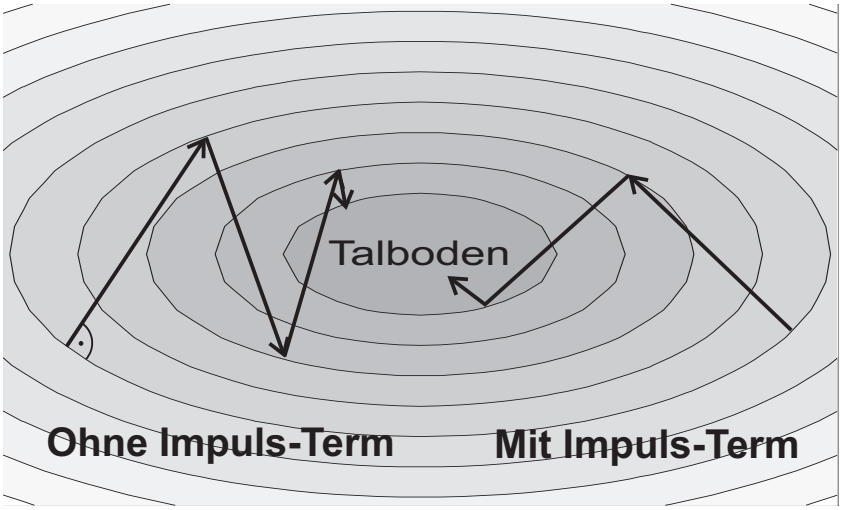


Abbildung 3: Gradienten-Abstieg ohne und mit Impuls-Term

Zusätzlich wird oft die Lernrate α während des Lernens verändert. Zum Beispiel wird die Lernrate verringert, wenn man erreichen möchte, dass man auch in relativ engen "Tälern" von $E(\mathbf{w})$ zum Talboden gelangt.

Anstelle von Backprop können auch andere Verfahren der numerischen Mathematik verwendet werden, die geeignet sind, schnell ein lokales Minimum der Funktion $E(\mathbf{w})$ zu finden (z.B. conjugate gradient, oder Levenberg-Marquart).

3 Was ist eigentlich "Lernen" (im Fall von Klassifikation und Regression)?

Angenommen, wir haben eine endliche Menge von Beispielen $\langle a, b\rangle$ für ein Klassifikations- oder Regressionsproblem, die wir in eine Liste L von l Trainingsbeispielen und eine Liste T von k Testbeispielen aufgeteilt haben. Nehmen wir ferner an, dass wir eine Familie $(\mathcal{H}_i)_{i\in\mathbb{N}}$ von Hypothesenklassen haben mit $\mathcal{H}_i \subseteq \mathcal{H}_{i+1}$ für alle i, sowie für jede Hypothesenklasse $\mathcal{H}_i$ einen Lernalgorithmus C_i, der nach dem Trainieren jeweils eine Hypothese aus $\mathcal{H}_i$ ausgibt.

Konkretes Beispiel: $\mathcal{H}_i$ besteht aus allen neuronalen Netzen der Tiefe 2 mit i Neuronen auf Schicht 1, und C_i ist der Backprop-Lernalgorithmus angewendet auf solche Netze.

Frage: Welches i sollen wir wählen, wenn unser Ziel ist, dass A_i angewendet auf L eine Hypothese $H_i := C_i(L)$ aus $\mathcal{H}_i$ ausgibt, sodass $error_T(H_i)$ (bzw. $MSE_T(H_i)$ im Fall eines Regressionsproblems) möglichst klein ist?

Frage: Weshalb wendet man nicht direkt C_i auf T anstatt L an, wenn unser Ziel ist, dass der Fehler der ausgegebenen Hypothese auf T möglichst klein sein soll?

Typische Situation:

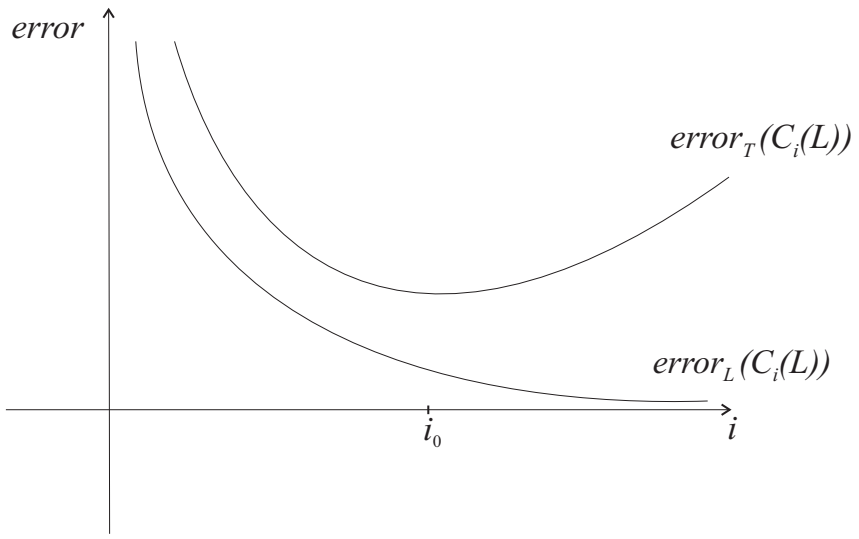


Abbildung 4: Typischer Verlauf der Fehler der Hypothese $C_i(L)$ auf Trainingsmenge L und Testmenge T in Abhängigkeit von der Größe der Hypothesenklasse $\mathcal{H}_i$. Die Hypothesenklasse $\mathcal{H}_{io}$ würde in diesem Fall das beste Ergebnis liefern.

Frage: Wie kann man intuitiv verstehen, dass $error_T(C_i(L))$ für große i wieder ansteigt?

Eine etwas vage Antwort: Je größer $\mathcal{H}_i$ ist, umso mehr Hypothesen gibt es in $\mathcal{H}_i$, die $error_L(H_i)$ klein machen, aber hohe Werte für $error_T(H_i)$ liefern.

Ein lehrreiches Beispiel: Regression mit Polynomen. Es sei $\mathcal{H}_i$ die Klasse aller Poly-

nome des Grades $\leq i$ in einer Variablen.

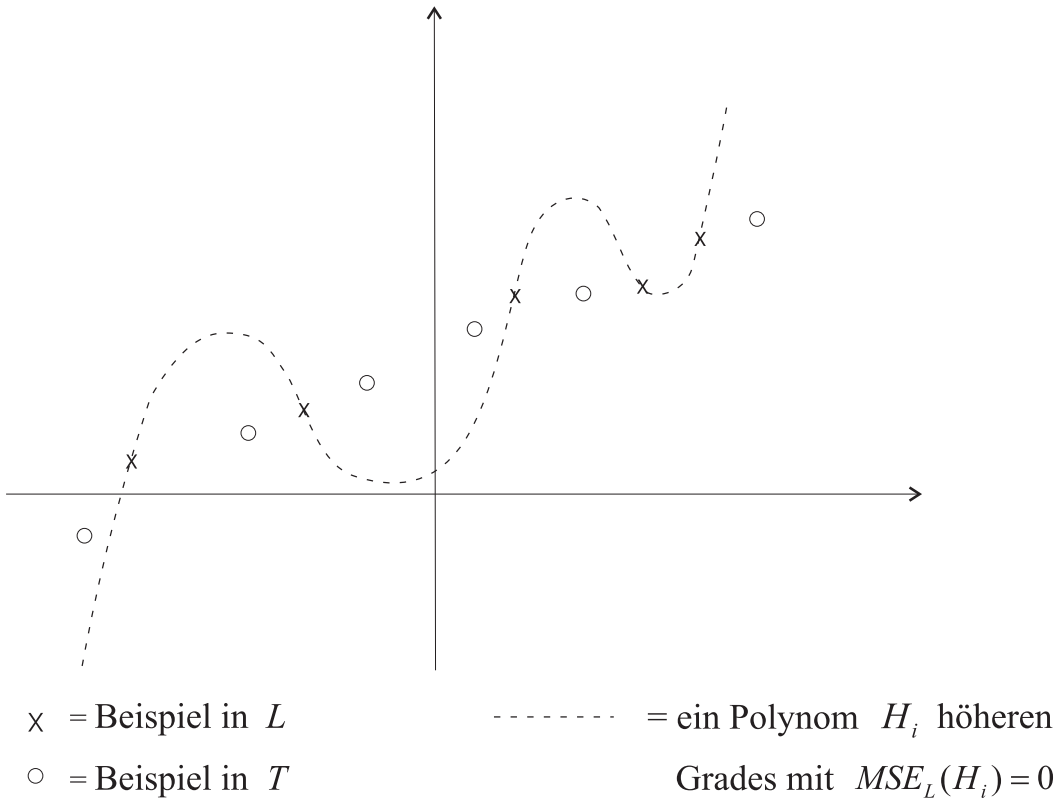


Abbildung 5: Illustration von möglichem Overfitting bei Verwendung eines Polynom höheren Grades (anstatt einer Geraden) bei einem Regressionsproblem

Frage: Was ist zu erwarten, wenn man $i = 1$ setzt und H_i aus $\mathcal{H}_i$ so gewählt wird, dass $MSE_L(H_i)$ minimiert wird?

Zu beachten: $\sup_{H \in \mathcal{H}_i} |MSE_L(H) - MSE_T(H)|$ wächst bei diesem Beispiel mit größer werdendem i stark an. Das ist sogar meist der Fall, nicht nur bei dem Beispiel mit $\mathcal{H}_i = \{$Polynome von Grad $\leq i\}$ oder dem Beispiel $\mathcal{H}_i = \{$neuronale Netze der Tiefe 2 mit i Neuronen auf Schicht 1$\}$. Daher kann man das größere Risiko beim Lernen mit Hypothesenklassen $\mathcal{H}_i$ für großes i so beschreiben: Wir können zwar in $\mathcal{H}_i$ mehr Hypothesen H_i finden mit geringem $MSE_L(H_i)$, aber gleichzeitig wächst mit i auch das Risiko, ein H_i zu erwischen, bei dem $MSE_L(H_i)$ und $MSE_T(H_i)$ weit auseinander klaffen.

Mögliche Nachteile der Verwendung von zu großen Hypothesenklassen hat man schon seit vielen Jahrhunderten in der Naturwissenschaft beobachtet: beim Bestreben quantitative Naturgesetze zu finden, die die Ergebnisse von neuen Beobachtungen möglichst gut vorhersagen.

Occam's Razor (von William von Occam, 14. Jahrhundert): Bevorzuge, wenn immer möglich, die einfachere Hypothese.

Albert Einsteins Regel: Ein Naturgesetz soll einfach sein, aber nicht zu einfach.

Minimum Description Length (MDL) Principle: Wähle eine Hypothese H zur Erklärung von Daten L so, dass die Summe

(minimale Bit-Länge der Beschreibung von H) + (minimale Bit-Länge der Beschreibung der Abweichung der Daten L von der Vorhersage H)

möglichst klein ist.

Welche Hypothese H gemäß dem letztgenannten Prinzip optimal ist, hängt natürlich ab von dem Formalismus den man benutzt, um Hypothesen H und Abweichungen von Daten L zu kodieren. Eine theoretisch optimale Kodierung wäre gemäß der Informationstheorie eine Kodierung, bei der

$$(\text{minimale Bit-Länge der Beschreibung von } H) = -\log_2 p(H)$$

und

$$(\text{minimale Bit-Länge der Beschreibung der Abweichung der Daten } L \text{ von der Vorhersage } H) = -\log_2 p(L|H)\,,$$

wobei $p(H)$ die Wahrscheinlichkeit des Auftretens von H ist. Ferner ist $p(L|H)$ die Wahrscheinlichkeit, dass die Daten L erzeugt werden, wenn H die "wahre" Hypothese ist. Diese Wahrscheinlichkeiten $p(H)$ und $p(L|H)$ (sowie $p(H|L)$) machen nur in der subjektiven Wahrscheinlichkeitstheorie von Bayes Sinn, bei der Wahrscheinlichkeiten nicht unbedingt nur Grenzwerte von relativen Häufigkeiten darstellen müssen, sondern auch einen subjektiven Glauben an die Richtigkeit von gewissen Annahmen quantifizieren können.[8] Das bekannte Theorem von Bayes

$$p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)}$$

gilt aber für beliebige Ereignisse A, B ($mit\ p(B) \neq 0$) auch in der üblichen Wahrscheinlichkeitstheorie.

Die im Minimal Description Length (MDL) Prinzip genannte Summe hat dann gemäß dem Theorem von Bayes den Wert

$$-\log_2 p(H) - \log_2 p(L|H) = -\log_2(p(H) \cdot p(L|H)) = -\log_2(L \cap H)$$

[8] Es ist dann $-\sum_H p(H) \cdot \log_2 p(H)$ die durchschnittliche zur Beschreibung der Hypothese H benötigte Anzahl von Bits. Dieser Term wird als die **Entropie** der Verteilung p bezeichnet. Gleichzeitig misst dieser Term die durchschnittliche Information (in bits), welche erforderlich ist, um das Ergebnis einer Ziehung einer Hypothese H gemäß der Verteilung p mitzuteilen.

$$= -\log_2(p(H|L) \cdot p(L)) = -\log_2 p(H|L) - \log_2 p(L)\,.$$

Wenn man für vorgegebene Daten L (also für fixes $\log_2 p(L)$) eine Hypothese wählt, die die im MDL-Prinzip auftretende Summe minimiert, so wählt man also "automatisch" eine Hypothese H, welche $p(H|L)$ maximiert. Das ist offensichtlich die optimale Wahl im Rahmen einer subjektiven Wahrscheinlichkeitstheorie. Man nennt ein solches H eine **maximum a posteriori (MAP)** Hypothese.

Das MDL-Prinzip ist von Interesse als abstraktes Ziel beim Entwurf von neuen Lernalgorithmen. Bei Anwendungen des maschinellen Lernens auf Klassifikationsprobleme, insbesondere bei der Wahl der richtigen Hypothesenklasse $\mathcal{H}_i$, hilft einem aber eher die Betrachtung der VC-Dimension dieser Hypothesenklassen, die wir im Abschnitt 3.1 kurz erläutern. Weitere praktisch anwendbare Methoden sind mittels sogenannter Regularisierungstheorien entwickelt worden, bei der die praktisch nicht berechenbare "minimale Bit-Länge einer Hypothese H" des MDL-Prinzips durch eine effizient berechenbare Zahl ersetzt wird, die auf andere Weise die "Komplexität" einer Hypothese H misst. Zum Beispiel kann dieser Term die Anzahl der Gewichte mit Wert $\neq 0$ in einer fixen großen neuronalen Netzwerk-Architektur messen, oder die Anzahl der Gatter in einem RBF-Netzwerk, oder die Nicht-Glattheit der durch das neuronale Netz berechneten Funktion. Man bezeichnet solch einen Term als **Regularisierungsterm**. Der zweite Term im MDL-Prinzip wird in einer Regularisierungstheorie ebenfalls durch einen praktisch berechenbaren Term ersetzt, der misst, wie gut die Hypothese an die Daten L angepasst ist (z.B. MSE von H auf L im Fall eines Regressionsproblems). In der so entstehenden Summe aus zwei Termen wird der Regularisierungsterm meist mit einem Faktor $\lambda > 0$ multipliziert, durch dessen Größe der Benutzer festlegen kann, wieviel Gewicht er auf die "Einfachheit" der Hypothese H und wieviel Gewicht er auf deren Angepasstheit an die Daten L legt.

Beachte: Erst durch einen Regularisierungsterm entsteht aus einem Interpolationsproblem ein Lernproblem. "Auswendig Lernen" verursacht in der Regel einen hohen Wert des Regularisierungsterms. Das entspricht der intuitiven Auffassung, dass Lernen nur dann stattfindet, wenn die in den Daten L vorhandene Information komprimiert wird, z.B. indem man die dahinter liegenden Prinzipien erfasst.

3.1 Die VC-Dimension einer Hypothesenklasse und das Uniforme Konvergenz Theorem

Man kann jeder Hypothesenklasse $\mathcal{H}$ für ein Klassifikationsproblem eine Zahl VC-dim($\mathcal{H}$) aus $\mathbb{N} \cup \{\infty\}$ zuordnen, die intuitiv gesprochen misst, wie viele Freiheitsgrade man hat bei der Wahl einer Hypothese H aus $\mathcal{H}$, oder genauer gesagt auf wie großen Mengen S von Beispielen $\langle a, b\rangle$ (mit beliebigen Target-Klassifikationen $b \in \{0, 1\}$) eine Hypothese H aus $\mathcal{H}$ gefunden werden kann, die 0 Fehler auf S macht.[9]

[9]Offensichtlich ist das nur möglich für **konsistente** Mengen S von Beispielen, für die es zu jedem a höchstens ein b mit $\langle a, b\rangle \in S$ gibt.

Definition der Vapnik-Chervonenkis Dimension (VC-Dimension)

Wir betrachten eine beliebige Klasse $\mathcal{H}$ von Funktionen $H : A \to \{0, 1\}$.

$$\text{VC-dim}(\mathcal{H}) := \max \{s \in \mathbb{N} : \text{es gibt eine Menge } S \subseteq A \text{ mit } s \text{ Elementen,}$$
$$\text{die von } \mathcal{H} \text{ zersplittert wird}\}.$$

Man sagt, dass eine Menge $S \subseteq A$ von $\mathcal{H}$ zersplittert wird, falls es zu jeder Funktion $f : S \to \{0, 1\}$ ein $H \in \mathcal{H}$ gibt, sodass $f(a) = H(a)$ für alle $a \in S$.

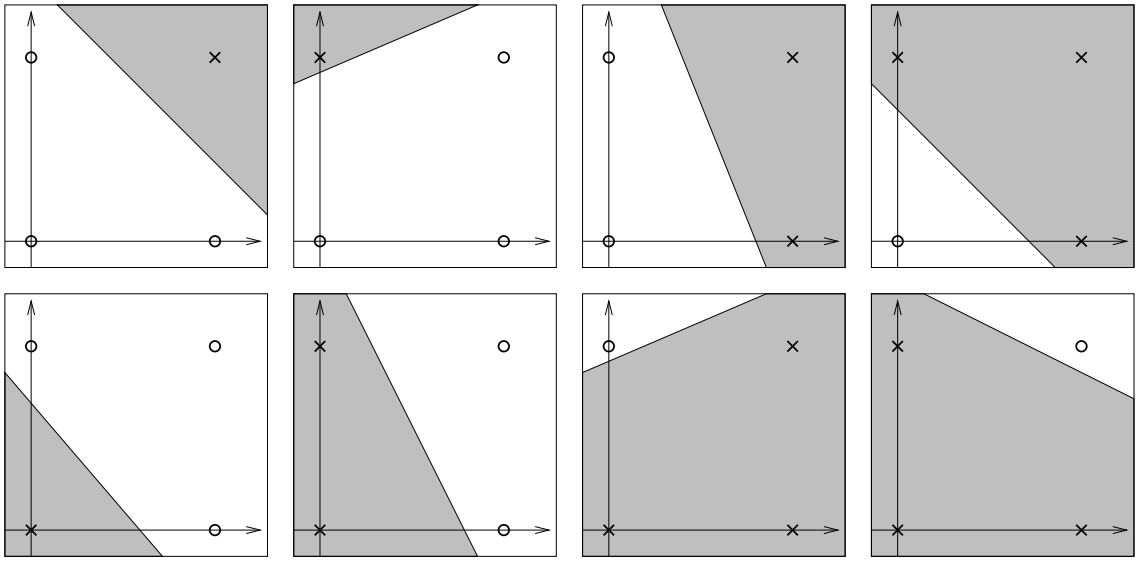


Abbildung 6: Nachweis, dass die Menge $S := \{\langle 0, 0\rangle, \langle 1, 0\rangle, \langle 0, 1\rangle\} \subseteq \mathbb{R}^2$ von der Hypothesenklasse $\mathcal{H}_{SG_2}$ (= Klasse aller Halbebenen H) zersplittert wird. Falls $H(x) = 1$ so ist der betreffende Gitterpunkt x jeweils mit einem Kreuz markiert.

Aus der Definition folgt sofort das VC-dim$(\mathcal{H}) \leq \log_2 |\mathcal{H}|$ für jede endliche Hypothesenklasse $\mathcal{H}$ bestehend aus $|\mathcal{H}|$ Hypothesen. Aber auch die unendlich große durch Schwellengatter über $\mathbb{R}^2$ definierte Hypothesenklasse $\mathcal{H}_{SG_2}$ hat eine endliche VC-Dimension: VC-dim$(\mathcal{H}_{SG_2}) = 3$. (In Abbildung 6 wird gezeigt, dass eine konkrete Menge S bestehend aus den 3 Punkten $\langle 0, 0\rangle$, $\langle 0, 1\rangle$, $\langle 1, 0\rangle$ im $\mathbb{R}^2$ von $\mathcal{H}_{SG_2}$ zersplittert werden kann, woraus VC-dim$(\mathcal{H}_{SG_2}) \geq 3$ folgt). Mit einem anderen Argument kann man beweisen, dass VC-dim$(\mathcal{H}_{SG_2}) < 4$; also VC-dim$(\mathcal{H}_{SG_2}) = 3$. Es ist in diesem Fall die VC-Dimension gleich der Anzahl der Parameter oder Freiheitsgrade, die man hat, um eine konkrete Hypothese $H \in \mathcal{H}_{SG_2}$ festzulegen. Das gilt auch für die Klasse $\mathcal{H}_{SG_d}$ der durch Schwellengatter über $\mathbb{R}^d$ definierten Hypothesen für beliebige $d \geq 1$: man hat hier stets VC-dim$(\mathcal{H}_{SG_d}) = d + 1$.

Es kann aber die VC-Dimension von mehrschichtigen Schwellenschaltkreisen sogar größer sein als die Anzahl der darin auftretenden Gewichte.[10] Die einfache Heuristik

[10] W. Maass. Neural nets with superlinear VC-dimension. Neural Computation, 6:877-884, 1994. http://www.igi.tugraz.at/maass/publications.html

VC-Dimension ($\mathcal{H}$) = (Anzahl der freien Parameter bei der Wahl einer Hypothese aus $\mathcal{H}$) stimmt also nicht für alle Hypothesenklassen $\mathcal{H}$.

Beispiele und Abschätzungen der VC-Dimension für neuronale Netze findet man in [Bartlett, P./Maass, W., 2003][11] (online erhältlich als # 139 auf `http://www.igi.tugraz.at/maass/publications.html`).

In Abhängigkeit von dieser Größe VC-dim($\mathcal{H}$) kann man nun abschätzen, wie viele Beispiele in der Trainingsmenge L sein müssen, damit beim Lernen mit Hypothesenklasse $\mathcal{H}$ die Wahrscheinlichkeit gering gehalten werden kann, dass $error_L(H)$ und $error_P(H)$ stark voneinander abweichen für die vom Lernalgorithmus aus $\mathcal{H}$ ausgewählte Hypothese H. Man geht hierbei von der pessimistischen Annahme aus, dass man nur dann zuversichtlich sein kann, dass die vom Lernalgorithmus $\mathcal{A}_i$ ausgegebene Hypothese $H_i = \mathcal{A}_i(L) \in \mathcal{H}_i$ nicht nur auf L, sondern auch auf einer genügend großen Testmenge T einen geringen Fehler hat (man sagt dann "H generalizes well"), falls $\sup_{H \in \mathcal{H}_i} |error_L(H) - error_P(H)|$ klein ist.

Uniformes Konvergenz Theorem:

Es sei P ein beliebiges Wahrscheinlichkeitsmaß auf $A \times B$ (mit $B = \{0,1\}$) und $\mathcal{H}$ sei eine beliebige (endlich oder unendlich große) Menge von Hypothesen $H : A \to B$. Wenn man nun für beliebige ε, $\delta > 0$ eine Menge L von mindestens

$$\frac{1}{\varepsilon^2}(constant \cdot VC\text{-}\dim(\mathcal{H}) + \ln\frac{1}{\varepsilon} + \ln\frac{1}{\delta})$$

Trainingsbeispiele gemäß P zieht, so gilt mit Wahrscheinlichkeit $\geq 1 - \delta$, dass

$$\sup_{H \in \mathcal{H}} |error_L(H) - error_P(H)| \leq \varepsilon \ .$$

Dabei ist *"constant"* eine Konstante > 0, von der angenommen wird, dass sie einen Wert nahe bei 1 hat (man konnte dies aber noch nicht beweisen).

Der Beweis dieses Theorems ist recht aufwendig. Die wesentliche Eigenschaft der VC-Dimension, die man hier ausnutzt, ist, dass für **jede** Teilmenge X von A mit einer beliebigen Anzahl m von Elementen höchstens $m^{\text{VC-}\dim(\mathcal{H})}$ der 2^m Funktionen von X nach $\{0,1\}$ durch Hypothesen $H \in \mathcal{H}$ definiert werden können.[12] In anderen Worten: die Anzahl der durch $\mathcal{H}$ induzierten Funktionen von X nach $\{0,1\}$ wächst nur polynomiell (anstatt exponentiell) mit der Anzahl m der Elemente der Teilmenge X von A.

[11] Peter L. Bartlett and W. Maass. Vapnik-Chervonenkis Dimension of Neural Nets. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1188-1192. MIT Press (Cambridge), 2nd edition, 2003.

[12] Die genaue Abschätzung ist $\left(\frac{e \cdot m}{\text{VC-}\dim(\mathcal{H})}\right)^{\text{VC-}\dim(\mathcal{H})}$. Für VC-dim($\mathcal{H}$) ≥ 3 ist dies $\leq m^{\text{VC-}\dim(\mathcal{H})}$.

3.2 Praktische Überlegungen zur Wahl der Hypothesenklasse, Trainings- und Testmenge

Die VC-Dimension einer Hypothesenklasse $\mathcal{H}_i$ kann oft recht grob abgeschätzt werden durch die Anzahl der Parameter deren Werte bei der Wahl einer bestimmten Hypothese $H \in \mathcal{H}_i$ festgelegt werden müssen, also zum Beispiel mit der Anzahl der Gewichte eines neuronalen Netzes. Das Uniforme Konvergenz Theorem verlangt, dass die Größe der Trainingsmenge mindestens so groß wie die VC-Dimension sein sollte. Das ist eine recht nützliche "Daumenregel".

Im Grunde müsste man ja gemäß dem Uniformen Konvergenz Theorem sogar noch eine viel größere Trainingsmenge haben. In der Regel riskiert man aber, diese Forderung nicht zu erfüllen. Man kann die Größe dieses Risikos mit Hilfe einer validation set V, die aus der Trainingsmenge L gezogen wird, selbst abschätzen. Bezeichnen wir diejenigen Trainingsbeispiele aus L die nicht in V kommen mit $\tilde{L}$. Dann liegt $error_V(H_i)$ mit großer Wahrscheinlichkeit nahe bei $error_P(H_i)$ für die vom Lernalgorithmus C_i aus $\mathcal{H}_i$ ausgewählte Hypothese $H_i := C_i(\tilde{L})$. Man vergleicht also einfach $error_V(H_i)$ mit $error_{\tilde{L}}(H_i)$. Falls $error_V(H_i)$ beträchtlich größer ist als $error_{\tilde{L}}(H_i)$ bezeichnet man dies als **Overfitting**, und iteriert das Ganze für eine kleinere Hypothesenklasse $\mathcal{H}_{i'}$ mit $i' < i$. Man nimmt in diesem Fall an, dass man sich im rechten Teil von Abbildung 4 befindet. Andernfalls, wenn $error_V(H_i)$ nicht viel größer als $error_{\tilde{L}}(H_i)$ ist, aber $error_V(H_i)$ noch zu hoch ist (**Underfitting**), kann man es sogar wagen, zu einer größeren Hypothesenklasse $\mathcal{H}_{i'}$ mit $i' > i$ zu gehen, in der Hoffnung, dass $error_T(H_{i'}) < error_V(H_i)$.

Wir nutzen hier aus, dass gemäß der Tschebyschev-Ungleichung und der Chernoff-Bound für genügend große validation sets die Wahrscheinlichkeit recht klein ist, dass $error_V(H)$ und $error_P(H)$ stark voneinander abweichen (siehe Abschnitt 1.2).

Meist hat man nur eine fixe Anzahl m von Beispielen für ein Klassifizierungsproblem, und man möchte daraus gerne sowohl eine Trainingsmenge, als auch ein validation set mit mehr als $m/2$ Beispielen generieren. Das wird mittels k**-fold Crossvalidation** für ein beliebig fixiertes $k \in \mathbb{N}$ mit $2 \leq k \leq m$ "fast" erreicht. Man teilt bei diesem Verfahren die Liste L aller Beispiele zufällig in k disjunkte Teilfolgen $L_1, \ldots, L_k$ auf. Sei $L^{(j)}$ die Liste, die aus allen Beispielen in L **außer** denen in L_j besteht. Man berechnet dann mit dem gewählten Lernalgorithmus C_i die Hypothese $C_i(L^{(j)}) \in \mathcal{H}_i$ für diese Teilmenge $L^{(j)}$, bestehend aus $m - \frac{m}{k}$ Beispielen, und testet sie auf dem validation set L_j, bestehend aus $\frac{m}{k}$ Beispielen. Da dieses validation set für große k in der Regel zu klein ist, iteriert man das Ganze für $j = 1, \ldots, k$, um besser abschätzen zu können, ob man $\mathcal{H}_i$ zu groß (Overfitting) oder zu klein (Underfitting) gewählt hat. Man wählt dann i so, dass $\frac{1}{k} \sum_{j=1}^{k} error_{L_j}(C_i(L^{(j)}))$ minimiert wird (in der Erwartung, dass dieser Durchschnitt eine genügend gute Abschätzung für $\frac{1}{k} \sum_{j=1}^{k} error_P(C_i(L^{(j)}))$ ergibt, oder sogar für $error_P(C_i(L))$ mit $L = L_1 \cup \ldots \cup L_k$).

Im Extremfall $k = m$, bei dem jedes L_j nur aus einem Beispiel besteht, und daher jede Hypothese $C_i(L^{(j)})$ nur für ein einziges Beispiel getestet wird, bezeichnet man dieses

Verfahren als **Leave-One-Out Crossvalidation**. Das ist eine Standardmethode, die man anwendet, wenn nur ein paar Hundert Beispiele (oder sogar noch weniger Beispiele) zur Verfügung stehen.

4 Wichtige Algorithmen zur Lösung von Klassifikationsproblemen

Wir beschreiben hier kurz drei Lernalgorithmen für Klassifikationsprobleme, die derzeit als die erfolgreichsten betrachtet werden. Welchen dieser Algorithmen man wählt, hängt von der zur Verfügung stehenden Rechenzeit beim Trainieren bzw. Anwenden, sowie von dem Wert, den man auf eine intuitive Interpretierbarkeit der ausgegebenen Hypothese legt, ab.

4.1 Der Nearest-Neighbor Algorithmus

Wir betrachten ein Klassifikationsproblem, gegeben durch ein WM P auf einer Menge $A \times B$ mit $A = X_1 \times \ldots \times X_d$. Für je zwei Beispiele $\langle\langle x_1, \ldots, x_d\rangle, b\rangle$, $\langle\langle \tilde{x}_1, \ldots, \tilde{x}_d\rangle, \tilde{b}\rangle$ aus $A \times B$ kann man den Abstand der beiden Attributvektoren zum Beispiel durch den folgenden Term messen:

$$D(\langle x_1, \ldots, x_d\rangle, \langle \tilde{x}_1, \ldots, \tilde{x}_d\rangle) := \sqrt{\sum_{i=1}^{d} d(x_i, \tilde{x}_i)}$$

mit

$$d(x_i, \tilde{x}_i) := \begin{cases} (x_i - \tilde{x}_i)^2 & , \text{ falls } |X_i| = \infty \text{ und } X_i \subseteq \mathbb{R} \\ 0 & , \text{ falls } |X_i| < \infty \text{ und } x_i = \tilde{x}_i \\ 1 & , \text{ falls } |X_i| < \infty \text{ und } x_i \neq \tilde{x}_i \ . \end{cases}$$

Der Nearest-Neighbor Algorithmus gibt für eine Trainingsmenge L die folgende Hypothese $H : A \to B$ aus:

$$H(\langle x_1, \ldots, x_d\rangle) \text{ ist die Klasse } \tilde{b} \text{ von demjenigen Beispiel } \langle\langle \tilde{x}_1, \ldots, \tilde{x}_d\rangle, \tilde{b}\rangle$$

$$\text{in } L, \text{ für das } D(\langle x_1, \ldots, x_d\rangle, \langle \tilde{x}_1, \ldots, \tilde{x}_d\rangle) \text{ minimal ist.}$$

Diese Form des Algorithmus wird oft mit IB1 bezeichnet. In einer IBk genannten Variante (für eine beliebige ungerade natürliche Zahl k) sucht man zu $\langle x_1, \ldots, x_d\rangle$ diejenigen k Beispiele $\langle\langle \tilde{x}_1, \ldots, \tilde{x}_d\rangle, \tilde{b}\rangle$ in L für die $D(\langle x_1, \ldots, x_d\rangle, \langle \tilde{x}_1, \ldots, \tilde{x}_d\rangle)$ minimal ist, und definiert $H(\langle x_1, \ldots, x_d\rangle)$ als die unter diesen k Beispielen am häufigsten auftretende Klasse $\tilde{b}$.

Vorteile dieses Algorithmus:

1. Die Komplexität der Hypothese wird automatisch an die Komplexität der Trainingsmenge angepasst.
2. Es können laufend weitere Beispiele in die Hypothese eingebaut werden, oder auch andere Beispiele entfernt werden.

Nachteile dieses Algorithmus:

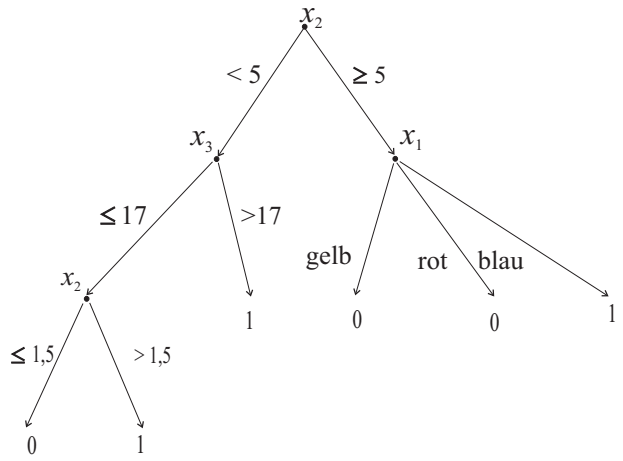
1. Bei der Anwendung der Hypothese zur Klassifikation neuer Beispiele wird relativ große Rechenzeit benötigt (falls L sehr groß ist). In einer IB2 genannten Variante des Algorithmus werden daher nur diejenigen Beispiele aus L zur Klassifikation neuer Beispiele verwendet, die nicht schon mittels der vorhergehenden Beispiele in L richtig klassifiziert werden konnten. Das kann aber dahin führen, dass fehlerhafte Beispiele besonders großen Einfluss gewinnen.

2. Falls einige der Attribute x_i in Wirklichkeit irrelevant sind für die Klassifikation, so schwächen sie die Leistung von diesem Lernalgorithmus.

3. Die numerischen Attribute müssen vorher vom Benutzer geeignet skaliert werden.

4.2 Lernen mit Entscheidungsbäumen

Falls zum Beispiel $A = \{\text{gelb, rot, blau}\} \times \mathbb{R}^2$ und $B = \{0, 1\}$, so definiert der folgende Entscheidungsbaum eine Hypothese $H : A \to B$ (wobei der Attributvektor aus A mit $\langle x_1, x_2, x_3 \rangle$ bezeichnet wird):



Wir besprechen im Folgenden den Lernalgorithmus C4.5. Dieser erzeugt für beliebige Trainingsmengen zu einem Klassifikationsproblem, gegeben durch ein WM P auf einer Menge $A \times B$ mit $|B| < \infty$, einen dazu passenden Entscheidungsbaum.[13] Dabei wird vorausgesetzt, dass A die Form $X_1 \times \ldots \times X_d$ hat, mit $X_i = \mathbb{R}$ oder $|X_i| < \infty$ für $i = 1, \ldots, d$.

Def.: Ein Entscheidungsbaum (EB) für Beispiele aus $A \times B$ mit $A = X_1 \times \ldots \times X_d$ und $|B| < \infty$ ist ein Baum[14], dessen Knoten und Kanten in folgender Weise bezeichnet sind:

[13] Wir schreiben $|B|$ für die Anzahl der Elemente einer Menge B und auch $|L|$ für die Anzahl der Elemente einer Folge L.

[14] Ein **Baum** ist in der Mathematik ein endlicher gerichteter Graph mit den Eigenschaften

1. Jedes Blatt ist mit einem Element $b \in B$ bezeichnet.

2. Jeder innere Knoten ist mit einem Attribut x_i ($i \in \{1, \ldots, d\}$) bezeichnet.

3. Für einen mit einem Attribut x_i bezeichneten Knoten sind die von diesem Knoten ausgehenden Kanten mit disjunkten Teilmengen $M_1, \ldots, M_k$ des Wertebereichs[15] X_i für dieses Attribut x_i bezeichnet, sodass $X_i = \bigcup_{j=1}^{k} M_j$.

Jeder von C4.5 erzeugte EB hat zusätzlich die Eigenschaften, dass

- im Fall $|X_i| < \infty$ die von einem mit x_i bezeichneten Knoten ausgehenden Kanten mit Mengen $M_j \subseteq X_i \cup \{\text{missing value}\}$ bezeichnet sind sodass $|M_j| = 1$
- im Fall $|X_i| = \infty$ die von einem mit x_i bezeichneten Knoten ausgehenden Kanten mit Mengen M_1, M_2 bezeichnet sind die Intervalle von $\mathbb{R}$ darstellen, sowie möglicherweise mit $M_3 = \{\text{missing value}\}$.

Frage: Zu welchen Folgen L von Beispielen aus $A \times B$ gibt es einen EB H, sodass $error_L(H) = 0$?

Frage: Was kann man von einem Lernalgorithmus erwarten, der nach Möglichkeit zu jeder Folge L einen EB H mit $error_L(H) = 0$ erzeugt?

Wir nehmen im Folgenden an, dass eine Liste L von Trainingsbeispielen gegeben ist. Der Lernalgorithmus C4.5 erzeugt aus L in zwei Phasen einen EB:

1. Phase: Erzeugung eines EB's H', sodass $error_L(H')$ möglichst klein ist, wobei gleichzeitig auch darauf geschaut wird, dass H' möglichst geringe Tiefe hat.

2. Phase: Zurückschneiden von H' ("Pruning").

Skizze der 1. Phase:[16] Der Entscheidungsbaum wird schrittweise, beginnend bei der Wurzel w, definiert. Für einen Knoten v des schon entwickelten Baumes schreiben wir L_v für die Teilfolge derjenigen Beispiele in der Trainingsfolge L, die zu diesem Knoten hinführen. Falls alle Beispiele in L_v zur selben Klasse b gehören, macht man v zu einem Blatt des EB mit Bezeichnung b. Andernfalls wird ein sogenanntes **splitting criterion** angewendet, um zu entscheiden, mit welchem Attribut x_i der Knoten v bezeichnet wird, und im Fall $X_i = \mathbb{R}$ welche Mengen M_j die von v ausgehenden Kanten bezeichnen sollen.

---

1. Es gibt genau einen Knoten, in dem keine Kante endet (die "Wurzel" des Baumes).

2. In jedem von der Wurzel verschiedenen Knoten endet genau eine Kante.

3. Jeder Knoten ist von der Wurzel auf einem gerichteten Pfad erreichbar.

Ein Knoten eines Baumes wird als **innerer Knoten** bezeichnet, falls von ihm eine Kante ausgeht, andernfalls als **Blatt**. Die **Tiefe** eines Baumes ist die maximale Länge eines gerichteten Pfades darin.

[15] Oft wird der Wertebereich um ein zusätzliches Element "missing value" ergänzt, also z.B. $X_2 = \mathbb{R} \cup \{\text{missing value}\}$.

[16] Mehr Details findet man unter `http://yoda.cis.temple.edu:8080/UGAIWWW/lectures/C45/`

Das Ziel ist, x_i so zu wählen, dass durch die Abfragung dieses Attributs die Entropie der Klassen in L_v möglichst stark reduziert wird. Diese Entropie erreicht den geringst möglichsten Wert 0 genau dann, wenn alle Beispiele in L_v zur selben Klasse gehören (dann kann man den Knoten zu einem Blatt des EB machen). Diese Entropie ist maximal, wenn jede Klasse in L_v gleich häufig auftritt.

Die Entropie der Klassen in L_v wird quantifiziert durch den Term

$$\text{Entropie}(L_v) := -\sum_{b \in B} p_{L_v}(b) \cdot \log_2 p_{L_v}(b) \ ,$$

wobei

$$p_{L_v}(b) := \frac{\text{Anzahl der Elemente in } L_v \text{ aus der Klasse } b}{\text{Anzahl aller Elemente in der Folge } L_v} \ .$$

Die durchschnittliche Reduktion der Entropie durch Abfragung des Attributs x_i an Knoten v, mit unmittelbar darunter entstehenden neuen Knoten $v_1, \ldots, v_m$, kann demnach durch den Term

$$\text{Entropie}(L_v) - \sum_{j=1}^{m} \frac{|L_{v_j}|}{|L_v|} \text{Entropie}(L_{v_j})$$

quantifiziert werden. Im Wesentlichen wählt C4.5 die Bezeichnung x_i für Knoten v (und die Bezeichnungen der Kanten, die von v ausgehen) so, dass diese Reduktion der Entropie am Knoten v maximal ist.[17]

Skizze der 2. Phase: C4.5 geht zu jedem Knoten v des in der 1. Phase konstruierten Baumes, der kein Blatt ist, angefangen bei denjenigen Knoten, deren direkte Nachfolger alle Blätter sind, und prüft (mittels eines heuristischen Kriteriums), ob die am Knoten v durchgeführte Spaltung der Folge L_v statistisch signifikant ist, d.h. durch genügend viele Beispiele in L_v gestützt wird. Falls sie nicht statistisch signifikant ist, wird die Verzweigung bei Knoten v wieder eliminiert, und v wird zu einem Blatt gemacht, mit der am häufigsten in L_v vorkommenden Klasse als Bezeichnung.

Vorteile des Lernalgorithmus C4.5:

1. Intuitive Interpretierbarkeit der von C4.5 ausgegebenen Hypothese (zum Beispiel im Vergleich der durch Backprop für neuronale Netze errechneten Hypothese), falls der ausgegebene EB nicht zu groß ist.

[17] Um zu vermeiden, dass der Algorithmus Attribute x_i mit einer großen Zahl m von diskreten Werten (z.B. x_i = "Geburtsdatum") übermäßig bevorzugt, wählt C4.5 in Wirklichkeit dasjenige x_i, welches den Quotienten

$$\frac{\text{Entropie}(L_v) - \sum_{j=1}^{m} \frac{|L_{v_j}|}{|L_v|} \text{Entropie}(L_{v_j})}{-\sum_{j=1}^{m} \frac{|L_{v_j}|}{|L_v|} \cdot \log_2 \frac{|L_{v_j}|}{|L_v|}}$$

maximiert. Der Ausdruck im Nenner ist die Entropie der Verteilung von L_v auf $L_{v_1}, \ldots, L_{v_m}$. Falls zum Beispiel alle L_{v_j} gleich groß sind, hat diese Entropie den Wert $\log_2 m$.

2. Durch das Pruning wird die Komplexität der Hypothese automatisch an die Komplexität der Trainingsmenge angepasst, um Overfitting zu vermeiden.

3. Falls ein Klassifikationsproblem sehr viele Attribute hat, werden oft nur einige dieser Attribute im von C4.5 ausgegebenen EB abgefragt. Man bekommt also zusätzliche Nebeninformation darüber, welche Attribute am wichtigsten sind für die Klassifikation.

4. C4.5 benötigt relativ wenig Rechenzeit.

5. Man kann die resultierende Hypothese (Entscheidungsbaum) auch in der Form von Regeln formulieren.

Gelegentliche Nachteile von C4.5:

1. Das Pruning ist oft nicht radikal genug. Für einige Standard Klassifikationsprobleme wurde gezeigt, dass es EBe der Tiefe 1 oder 2 mit relativ kleinen wahren Fehlern gibt, während C4.5 für dieselben Klassifikationsprobleme recht große EBe ausgibt, die kaum noch intuitiv interpretierbar sind, und auch keinen wesentlich geringeren wahren Fehler haben.

2. Falls man viele Attribute mit kontinuierlichen Werten hat, erzielt ein Lernalgorithmus wie Backprop oft bessere Ergebnisse (weshalb?).

4.3 Support Vector Machines (SVMs)

Varianten des in diesem Abschnitt vorgestellten Lernalgorithmus erzielen gegenwärtig bei Klassifikationsproblemen in der Regel die besten Ergebnisse. Allerdings erfordert dieser Lernalgorithmus beträchtliche Rechenzeit.

Wir greifen zunächst einen in Abschnitt 2 diskutierten Ansatz wieder auf: Lernen mit einem Perzeptron, d.h. mit dem einfachst möglichen neuronalen Netz. Hierbei verwendeten wir Hypothesen der Form

$$H_{\mathbf{w}} := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w} \cdot \mathbf{x} \geq 0\}$$

für WMe P auf $\mathbb{R}^d \times \{0,1\}$. Der große Nachteil dieser Hypothesenklasse ist, dass für viele praktisch auftretende Trainingsfolgen L für Klassifikationsprobleme die Mengen

$$\text{POS} := \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{x}, 1\rangle \text{ tritt in } L \text{ auf}\}$$

und

$$\text{NEG} := \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{x}, 0\rangle \text{ tritt in } L \text{ auf}\}$$

nicht **linear trennbar** sind, d.h. es gibt keine Hypothese $H_{\mathbf{w}}$, sodass $\mathbf{x} \in H_{\mathbf{w}}$ für alle $\mathbf{x} \in \text{POS}$ und $\mathbf{x} \notin H_{\mathbf{w}}$ für alle $\mathbf{x} \in \text{NEG}$. Schlimmer noch, es kommt häufig vor, dass jede Hypothese der Form $H_{\mathbf{w}}$ einen so großen Wert von $error_L(H_{\mathbf{w}})$ erzielt, dass sie praktisch

nicht brauchbar ist. Ferner kann man einen Gewichtsvektor $H_{\mathbf{w}}$, der $error_L(H_{\mathbf{w}})$ minimiert praktisch nicht berechnen, weil dieses Problem NP-vollständig ist.

Oft sind diese Mengen POS, NEG aber gut trennbar durch Hypothesen H, die mittels quadratischer Ausdrücke, oder allgemeiner Polynome eines endlichen Grades p, definiert sind. Nehmen wir zum Beispiel an, dass POS, NEG $\subseteq \mathbb{R}^d$ durch eine quadratische Menge der Form

$$\{\mathbf{x} \in \mathbb{R}^d : \sum_{1 \leq i \leq j \leq d} w_{ij} x_i x_j + \sum_{i=1}^{d} w_i x_i + w_0 \geq 0\}$$

getrennt werden können. Betrachten wir die Projektion $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ für $d' := \begin{pmatrix} d \\ 2 \end{pmatrix} + d$ definiert durch $\Phi(\mathbf{x}) = \langle x_1, \ldots, x_d, x_1 \cdot x_1, \ldots, x_i \cdot x_j, \ldots, x_d \cdot x_d \rangle$. Dann sind die durch diese Projektion Φ in den höher-dimensionalen Raum $\mathbb{R}^{d'}$ projizierten Mengen[18] Φ[POS], Φ[NEG] linear trennbar.

Dieser Effekt wird beim Lernen mit Support Vector Machines massiv ausgenutzt: es zeigt sich, dass für fast alle in der Praxis auftretenden Klassifikationsprobleme P bzw. Trainingsfolgen L die Mengen Φ[POS], Φ[NEG] linear trennbar (oder fast linear trennbar) sind für geeignete **nichtlineare** Projektionen Φ der Attributvektoren $\mathbf{x}$ in genügend hochdimensionale Räume $\mathbb{R}^{d'}$.

Frage: Wieso sind **lineare** Projektionen Φ wenig aussichtsreich?

Diese Beobachtung erlaubt einem, das ursprüngliche Klassifikationsproblem im $\mathbb{R}^d$ auf ein Klassifikationsproblem im $\mathbb{R}^{d'}$ zurückzuführen, bei dem es reicht, lineare Hypothesen der Form $H_{\mathbf{w}}$ mit $\mathbf{w} \in \mathbb{R}^{d'}$ zu betrachten. Das löst aber noch nicht automatisch das durch das WM P auf $\mathbb{R}^d \times \{0,1\}$ definierte Lernproblem, denn wenn d' sehr groß ist, muss man mit Overfitting rechnen. Aus diesem Grund reduziert man die Vielfalt der in Frage kommenden Hypothesen $H_{\mathbf{w}}$ mit $\mathbf{w} \in \mathbb{R}^{d'}$ durch eine zusätzliche Forderung: Man verlangt nicht nur, dass $H_{\mathbf{w}}$ die Mengen Φ[POS] und Φ[NEG] trennt, sondern dass die durch $H_{\mathbf{w}}$ definierte trennende Hyperebene $\{\mathbf{x} \in \mathbb{R}^{d'} : H_{\mathbf{w}}(\mathbf{x}) = 0\}$ einen möglichst großen Abstand (margin) von den beiden Mengen Φ[POS], Φ[NEG] erzielt. Das schränkt die in Frage kommenden Hypothesen drastisch ein, denn während es bei linear trennbaren Mengen Φ[POS], Φ[NEG] stets unendlich viele trennende Hyperebenen $H_{\mathbf{w}}$ gibt, gibt es nur genau eine trennende Hyperebene H_{opt}, für die der minimale Abstand zu Elementen von Φ[POS] $\cup$ Φ[NEG] einen maximalen Wert erreicht. Man nennt diese eindeutig bestimmte Hyperebene H_{opt} die **maximal margin Hyperebene** für das durch Φ[POS], Φ[NEG] definierte Klassifikationsproblem. H_{opt} hat die zusätzliche Eigenschaft, dass sie stets in der Form

$$H_{opt} = \{\underline{z} \in \mathbb{R}^{d'} : \sum_{i=1}^{k} \alpha_i \cdot \Phi(\mathbf{x}(i)) \cdot \underline{z} + \alpha_0 = 0\}$$

für endlich viele **support vectors** $\Phi(\mathbf{x}(i))$ mit $\mathbf{x}(i) \in$ POS $\cup$ NEG geschrieben werden

[18] Falls f eine Abbildung von einer Menge M in eine Menge S ist, und $X \subseteq M$, so schreiben wir $f[X]$ für $\{f(x) : x \in X\}$.

kann.[19] Diese Hyperebene H_{opt} definiert über dem Raum der Attributvektoren $\mathbf{x} \in \mathbb{R}^d$ den maximal margin classifier

$$MMC_{opt} := \{\mathbf{x} \in \mathbb{R}^d : \sum_{i=1}^{k} \alpha_i \Phi(\mathbf{x}(i)) \cdot \Phi(\mathbf{x}) + \alpha_0 \geq 0\}.$$

Die Koeffizienten α_i dieses maximal margin classifiers MMC_{opt} können durch Lösung eines konvexen quadratischen Optimierungsproblems explizit berechnet werden, was aber leider beträchtliche Rechenzeit erfordert. Es ist hier kein iteratives Verfahren wie bei der Perzeptron Lernregel (siehe Abschnitt 2.3) erforderlich. Das quadratische Optimierungsproblem hat bei einer Liste $L = \{\langle \mathbf{x}(1), b_1 \rangle, \ldots, \langle \mathbf{x}(m), b_m \rangle\}$ von Trainingsbeispielen die folgende Form:

$$\text{Maximiere} \qquad \sum_{i=1}^{m} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{m} \alpha_i \, \alpha_j \, b_i \, b_j \, \Phi(\mathbf{x}(i)) \cdot \Phi(\mathbf{x}(j))$$

unter den Nebenbedingungen

$$0 \leq \alpha_i \leq C, \; i = 1, \ldots, m \quad \text{und} \quad \sum_{i=1}^{m} \alpha_i \, b_i = 0.$$

Dabei ist $C > 0$ eine vom Benutzer zu wählende Konstante, die festlegt, wieviel Wert der Benutzer zum Einen auf die Minimierung der Anzahl der falsch klassifizierten Beispiele $\Phi(\mathbf{x}(i))$ legt (wenn ihm dies wichtig ist, wählt er einen großen Wert C), und zum Anderen auf die Maximierung des minimalen Abstands der trennenden Hyperebene von einem richtig klassifizierten Beispiel.

Für Anwendungen wesentlich ist die Wahl einer geeigneten Projektion $\Phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ des Klassifizierungsproblems in einen hoch-dimensionalen Raum $\mathbb{R}^{d'}$, sodass $\Phi[POS], \Phi[NEG]$ möglichst gut *linear* trennbar sind. Dabei nutzt man aus, dass man bei Verwendung einer beliebigen Projektion Φ für die Berechnung und Anwendung von H_{opt} lediglich Produkte der Form

$$k(\mathbf{x}, \mathbf{y}) := \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$$

für $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ ausrechnen muss. Auch der durch H_{opt} induzierte maximal margin classifier MMC_{opt} für die Attributvektoren $\mathbf{x} \in \mathbb{R}^d$ von Testbeispielen kann ohne Berechnung von Φ nur mittels dieser Funktion $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ definiert werden:

$$MMC_{opt} = \{\mathbf{x} \in \mathbb{R}^d : \sum_{i=1}^{k} \alpha_i \cdot k(\mathbf{x}(i), \mathbf{x}) + \alpha_0 \geq 0\} \; ,$$

wobei die Koeffizienten α_i durch die Lösung des genannten quadratischen Optimierungsproblems definiert sind.

[19] Oft sind die $\mathbf{x}(i)$ der in der Definition dieser Hypothese H implizit auftretenden support vectors $\Phi(\mathbf{x}(i))$ auch für das intuitive Verstehen des zu Grunde liegenden Klassifikationsproblems P nützlich, weil diese $\mathbf{x}(i)$ als charakteristische **Grenzfälle** interpretiert werden können.

Für viele wichtige Projektionen Φ kann man diese als **kernel** bezeichnete Funktion $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ sehr leicht ausrechnen. Im Fall, dass $\Phi(\mathbf{x})$ für ein beliebiges $p \in \mathbb{N}$ der Vektor bestehend aus allen Produkten des Grades $\leq p$ von Komponenten von $\mathbf{x}$ ist (sodass jedes Polynom vom Grad $\leq p$ in den Komponenten von $\mathbf{x}$ als gewichtete Summe von Komponenten von $\Phi(\mathbf{x})$ geschrieben werden kann), ist zum Beispiel

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p \ .$$

Aufgrund solcher "kernel-tricks" kann man sich sogar leisten, mit Projektionen Φ in unendlich-dimensionale Räumen zu arbeiten. Ein Beispiel ist der häufig verwendete RBF-kernel (RBF = radial basis function)

$$k(\mathbf{x}, \mathbf{y}) := \exp\left(\frac{- \parallel \mathbf{x} - \mathbf{y} \parallel^2}{2\sigma^2}\right)$$

dessen zugehörige Projektion $\Phi : \mathbb{R}^d \to \mathbb{R}^\infty$ die Eigenschaft hat, dass für beliebige, paarweis verschiedene Attributvektoren $\mathbf{x}(1), \ldots, \mathbf{x}(m)$ (mit beliebig großem $m \in \mathbb{N}$) die hochprojizierten Vektoren $\Phi(\mathbf{x}(1)), \ldots, \Phi(\mathbf{x}(m))$ stets linear unabhängig sind. Dies hat die attraktive Konsequenz, dass $\Phi[POS], \Phi[NEG]$ stets linear trennbar sind, falls die Trainingsfolge L widerspruchsfrei ist [weshalb?].

Nähere Informationen über die Wahl von kernels sowie geeignete Software findet man auf `http://www.kernel-machines.org/index.html`.

Abschließend möchten wir noch einmal die drei wesentlichen Ideen beim Design des SVM-Lernalgorithmus zusammenfassen:

1. Nichtlineare Projektion der Attribut-Vektoren in einen hochdimensionalen Raum, um die Trennbarkeit der Klassen durch lineare Hyperebenen zu verbessern.

2. Vermeidung des dadurch verursachten Overfitting Problems (aufgrund der i. A. sehr großen Zahl der freien Parameter einer Hyperebene im hochdimensionalen Raum) durch eine zusätzliche Anforderung an die gesuchte Hyperebene, welche die Zahl ihrer Freiheitsgrade reduziert: Wahl der maximal margin Hyperebene.

3. Vermeidung der durch Verwendung eines hochdimensionalen Raums zu erwartenden rechnerischen Probleme durch den kernel-trick, welcher die Berechnung der maximal margin Hyperebene im hochdimensionalen Raum auf die Lösung eines konvexen quadratischen Optimierungsproblems im Raum der ursprünglichen Attributvektoren reduziert.

5 Unüberwachtes Lernen

Wir betrachten nun den Fall, wo es keine "Bewertung" $b \in B$ von Attributsvektoren $a \in A$ mehr gibt, sondern nur noch Attributsvektoren. Wie kann man überhaupt etwas lernen, wenn man nur noch Beispiele der Form $a \in A$ bekommt (die gemäß einem WM P auf A gezogen werden)? Ein Beispiel hatten wir bei der Einführung am Anfang von Kapitel 1 kennengelernt ("lernen zu erkennen, wenn unser Auto ein ungewöhnliches Geräusch macht"): Novelty Detection.

Es gibt aber noch viele andere Anwendungen des unüberwachten Lernens, z.B.

- **Dimensionsreduktion**, d.h. man versucht, die Daten $a \in A$ effizienter darzustellen mit Attributsvektoren von geringerer Länge.

  Beispiele: Principal Component Analysis (PCA), Independent Component Analysis (ICA), Kohonen Maps.

- **Clustering** von Daten, auch in Verbindung mit der Erstellung von Prototypen für jedes Cluster. Dies wird oft (wie auch Dimensionsreduktion) zur Datenkompression verwendet.

  Beispiele: k-Means Algorithm, Learning Vector Quantization (LVQ).

- **Lernen von Assoziationen**
  Beispiele: Modellierung von Abhängigkeiten zwischen Zufallsvariablen (oft mittels Graphical Models), Google Suchmaschine, Hopfield-Netze

- **Vorhersage von Zeitreihen**
  Beispiel: Vorhersage von Aktienkursen, Wettervorhersage.

- **Blind Source Separation:** Entmischung von überlagerten Signalen.

5.1 Principal Component Analysis (PCA)

Wir nehmen hier an, dass P ein WM auf $A := \mathbb{R}^d$ ist mit $E_P(\mathbf{x}) = 0$. Die $d \times d$ Matrix $K := E[\mathbf{x} \cdot \mathbf{x}^T]$ wird dann als die Kovarianzmatrix der Verteilung P bezeichnet. Da K reellwertig und symmetrisch ist (weshalb?), gibt es eine reellwertige $d \times d$ Matrix $S = (\mathbf{s}, \ldots, \mathbf{s}_d)$ mit orthonormalen Spalten (d.h. $\mathbf{s}_i \cdot \mathbf{s}_j = 1$ falls $i = j$, sonst $= 0$) und reelle Zahlen ("Eigenwerte") $\lambda_1 \geq \ldots \geq \lambda_d \geq 0$, sodass

$$K\mathbf{s}_i = \lambda_i \mathbf{s}_i \quad \text{für} \quad i = 1, \ldots, d.$$

Die Spalten $\mathbf{s}_i$ der Matrix S sind also orthonormale Eigenvektoren zu den Eigenwerten λ_i. Sie werden auch als Hauptachsen (principal axes) der Matrix K bezeichnet.

Wir betrachten nun die linear transformierten Zufallsvariablen $\mathbf{y} := S^T \cdot \mathbf{x}$. Deren Kovarianzmatrix hat dann die Form $E[\mathbf{y} \cdot \mathbf{y}^T] = E[S^T \cdot \mathbf{x} \cdot (S^T \cdot \mathbf{x})^T] = E[S^T \cdot \mathbf{x} \cdot \mathbf{x}^T \cdot S] =$

$S^T \cdot E[\mathbf{x} \cdot \mathbf{x}^T] \cdot S = S^T \cdot K \cdot S = S^T \cdot (\lambda_1 \mathbf{s}_1, \ldots, \lambda_d \mathbf{s}_d) = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_d \end{pmatrix}$. Die Koordinaten $y_i = \mathbf{s}_i^T \cdot \mathbf{x}$ des transformierten Zufallsvektors $\mathbf{y} = S^T \cdot \mathbf{x}$ sind also unkorreliert (d.h. $E[y_i \cdot y_j] = 0$ für $i \neq j$), und $\lambda_i = E[y_i \cdot y_i]$ ist die Varianz von y_i.

Man kann nun mittels der Hauptachsen $\mathbf{s}_1, \ldots, \mathbf{s}_d$ für jedes $m \leq d$ eine lineare Projektion von $\mathbb{R}^d$ in den durch $\mathbf{s}_1, \ldots, \mathbf{s}_m$ aufgespannten m-dimensionalen Unterraum definieren: $\mathbf{x} \mapsto \langle (\mathbf{s}_1 \cdot \mathbf{x}) \cdot \mathbf{s}_1, \ldots, (\mathbf{s}_m \cdot \mathbf{x}) \cdot \mathbf{s}_m \rangle = \langle y_1 \cdot \mathbf{s}_1, \ldots, y_m \cdot \mathbf{s}_m \rangle$. Diese Projektion hat die folgende Extremaleigenschaft:

$$E[\|\mathbf{x} - \sum_{i=1}^{m} (\mathbf{s}_i \cdot \mathbf{x}) \cdot \mathbf{s}_i\|^2] \leq E[\|\mathbf{x} - \sum_{i=1}^{m} (\mathbf{z}_i \cdot \mathbf{x}) \cdot \mathbf{z}_i\|^2]$$

für beliebige orthonormale Vektoren $\mathbf{z}_1, \ldots, \mathbf{z}_m \in \mathbb{R}^d$.

In Worten: die Projektion von $\mathbf{x}$ auf die Hauptachsen $\mathbf{s}_1, \ldots, \mathbf{s}_m$ ist bzgl. MSE die bestmögliche Approximation des d-dimensionalen Zufallsvektors $\mathbf{x}$ durch einen m-dimensionalen Zufallsvektor.

Man bezeichnet die Projektion von $\mathbf{x}$ auf die Hauptachsen der Kovarianzmatrix K von $\mathbf{x}$ als *Principal Component Analysis* des Zufallsvektors $\mathbf{x}$. Dies ist die Standardmethode zur Dimensionsreduktion von hochdimensionalen Daten, zum Beispiel in der Bildverarbeitung und als Vorverarbeitung zur Klassifikation.

Falls $\lambda_d > 0$, so hat der durch $z_i := \frac{y_i}{\sqrt{\lambda_i}}$ definierte Zufallsvektor $\mathbf{z} \in \mathbb{R}$ die Eigenschaft, dass $E[\mathbf{z} \cdot \mathbf{z}^T] = I_d$ (= Matrix mit "Einsen" in der Diagonale, sonst nur "Nullen"). Man bezeichnet dann die Transformation $\mathbf{x} \mapsto \mathbf{z}$ als "whitening". Dies ist zum Beispiel eine geeignete Vorverarbeitung für den Nearest Neighbor Algorithmus.