Codierung

Auszug aus dem Skript von Maciej Liśkiewicz und Henning Fernau

1 Ein bisschen Informationstheorie

Betrachten wir das folgende Problem: Wie lautet eine sinnvolle Definition für das quantitative Maß der Information?

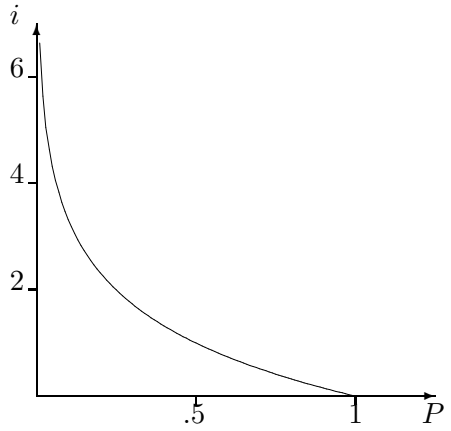


Abbildung 1: Die Selbstinformation in Abhängigkeit von $P(A)$

Elwood Shannon hat dieses Maß wie folgt definiert: Sei A ein Ereignis und $P(A)$ die Wahrscheinlichkeit, dass A eintritt; definiere dann

$$i(A) = \log_2 \frac{1}{P(A)}.$$

Shannon hat $i(A)$ *Selbstinformation* (engl.: self-information) genannt. Wir wollen $i(A)$ auch als *Informationsgehalt* ansprechen. Das Verhalten der Selbstinformation in Abhängigkeit von der Wahrscheinlichkeit P des Ereignisses A ist in Bild 1 grafisch dargestellt.

Intuitiv ist der Begriff klar: Seien A und B zwei Ereignisse (z. B. $A =$ „Der Verbrecher ist kleiner als 2m.“ und $B =$ „Der Verbrecher ist größer als 2m.“) mit den Wahrscheinlichkeiten

$$P(A) = 0,999 \quad \text{und} \quad P(B) = 0,001.$$

Eine Zeugenaussage, A treffe zu, ist keine große Überraschung, und deshalb enthält sie nicht viel Information ($i(A)$ ist sehr klein). Eine Nachricht, daß B wahr ist, hat dagegen einen großen Informationsgehalt.

Sei S eine bestimmte Informationsquelle. Die *Entropie* ist der „mittlere Informationsgehalt pro Nachricht aus S":

$$H(S) = \sum_{A \in S} P(A) \cdot i(A),$$

wobei wir für $P(A) = 0$ das Produkt $P(A) \cdot i(A)$ als 0 betrachten (dies ist sinnvoll, weil $\lim_{x \to 0} x \cdot \log \frac{1}{x} = 0$). Wir werden meistens S als endliches Quellenalphabet $\Sigma = \{a_1, \ldots, a_n\}$ betrachten. Dann ist $P(a_i)$ die Wahrscheinlichkeit, daß a_i eintritt und der Informationsgehalt $i(a_i) = \log_2 1/P(a_i)$ beschreibt die Anzahl von Bits (deshalb Basis 2 für den Logarithmus), die für die Kodierung des Zeichens a_i nötig ist (wobei theoretisch Bruchteile von Bits zugelassen werden). Der Einfachheit halber lassen wir im Folgenden die Basisangabe bei Logarithmen weg und treffen die Übereinkunft, 2 sei unsere Standardbasis. Die Entropie

$$H(\Sigma) = \sum_{i=1}^{n} P(a_i) \cdot \log \frac{1}{P(a_i)}$$

ist die mittlere Anzahl von Bits um eine Nachricht aus Σ zu codieren.

	$P(a)$	$P(b)$	$P(c)$	$P(d)$	$P(e)$	H
1.	$0,2$	$0,2$	$0,2$	$0,2$	$0,2$	$2,322$
2.	$0,5$	$0,25$	$0,125$	$0,0625$	$0,0625$	$1,875$
3.	$0,75$	$0,0625$	$0,0625$	$0,0625$	$0,0625$	$1,3$
4.	$0,94$	$0,01$	$0,01$	$0,01$	$0,01$	$0,322$

Tabelle 1: Beispiele für Entropien

Beispiele zur Entropie: Sei $\Sigma = \{a, b, c, d, e\}$ mit $P(a) = P(b) = P(c) = 0,25$ und $P(d) = P(e) = 0,125$. Dann ist

$$H = 3 \cdot 0,25 \cdot \log 4 + 2 \cdot 0,125 \cdot \log 8 = 1,5 + 0,75 = 2,25.$$

Weitere Beispiele enthält Tabelle 1.

Eine interessante Eigenschaft ist, daß die Entropie für ungleichmäßigere Verteilungen kleiner wird. Als weiteres Beispiel, das diese Eigenschaft illustriert, betrachten wir die *binäre Entropie-Funktion*:

$$H_b(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p},$$

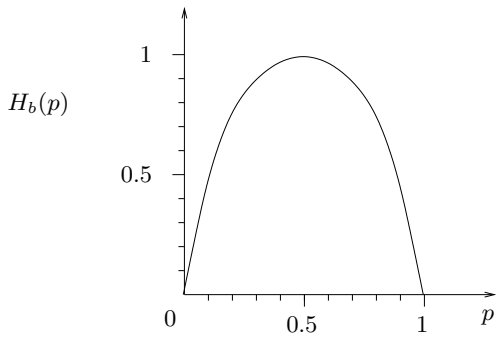


Abbildung 2: Die binäre Entropie-Funktion H_b

das heißt, die Entropie-Funktion für zwei Quellensymbole mit Wahrscheinlichkeiten p und $1-p$. Die Werte für $H_b(p)$, wobei p zwischen 0 und 1 variiert, zeigt das Diagramm 2.

2 Grundlegende Codes

2.1 Präfixcodes

Betrachten wir ein Codierungverfahren, das jedem Symbol aus dem Quellenalphabet genau ein binäres Codewort zuordnet. Die Codierung hat zusätzlich folgende Eigenschaft: Kein Codewort ist ein Präfix des anderen Codeworts. Die Codes, die diese Eigenschaft haben, nennen wir *Präfixcodes*. Mit Hilfe von Präfixcodes können wir eine Folge von Symbolen so codieren, dass wir einfach die Codewörter hintereinander schreiben, ohne zusätzliche Trennsymbole zu benutzen. Wie man sich leicht überlegt, führt die Einführung von Trennsymbolen nach Codewörtern dazu, einen Präfixcode zu erzeugen.

In diesem Kapitel zeigen wir den Satz von Kraft und beschreiben dann die grundlegenden Präfixcodes.

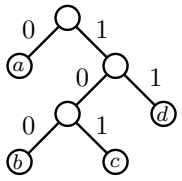


Abbildung 3: Der Baum eines Präfixcodes

Eine einfache Kennzeichnung von Präfixcodes ist die folgende: Im Präfixcode ist jedes Symbol durch ein Blatt in einem Binärbaum repräsentiert, und das Symbol wird durch den Weg von der Wurzel bis zum Blatt binärcodiert (links = 0, rechts = 1). Als Beispiel betrachte man Bild 3.

Satz 2.1 (Kraft 1949): *Es existiert ein Präfixcode* $K : \{a_1, \ldots, a_n\} \to \{0,1\}^+$ *mit* $|K(a_1)| = \ell_1, \ldots, |K(a_n)| = \ell_n$ *genau dann wenn*

$$\sum_{i=1}^{n} 2^{-\ell_i} \leq 1. \tag{1}$$

Die Relation (1) heißt auch *Kraftsche Ungleichung*.

Beweis: Nehmen wir an, es gibt einen Präfixcode $K : \Sigma \to \{0,1\}^+$, wobei $\Sigma = \{a_1, \ldots, a_n\}$ ein Quellenalphabet ist. Dann sei $\ell_i = |K(a_i)|$ für $i = 1, \ldots, n$ und es sei

$$m = \max\{\ell_1, \ldots, \ell_n\}.$$

Betrachten wir jetzt den Binärbaum T für K, das heißt T hat n Blätter und jedes Blatt repräsentiert ein Codewort. Jetzt verlängern wir T zu einem vollständigen Binärbaum T' der Höhe m. Ein Beispiel für $n = 6$, $m = 4$ und einen Code $K(a_i) = c_i$ findet sich in Bild 4.

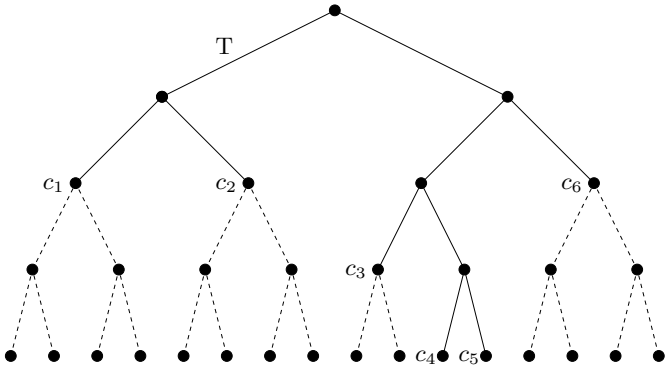


Abbildung 4: Ein (vervollständigter) Binärbaum T'

Jetzt betrachten wir die Teilbäume mit Wurzeln $c_1, c_2, \ldots, c_n$ und stellen fest daß die Teilbäume disjunkt sind. Es ist einfach zu sehen, daß ein Baum mit Wurzel c_i $2^{m-\ell_i}$ Blätter hat. Weil alle Teilbäume disjunkt sind und T' insgesamt 2^m Blätter hat, ergibt sich $\sum_{i=1}^{n} 2^{m-\ell_i} \leq 2^m$. Das impliziert die Kraftsche Ungleichung.

Jetzt nehmen wir an, dass die Kraftsche Ungleichung $\sum_{i=1}^{n} 2^{-\ell_i} \leq 1$ gilt. Wir wollen einen Code K konstruieren mit $|K(a_i)| = \ell_i$ für $i = 1, \ldots, n$.

Es sei $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_n = m$. Betrachten wir nun einen vollständigen Binärbaum T der Höhe m, wie beispielweise in Bild 5. Um den Code K zu

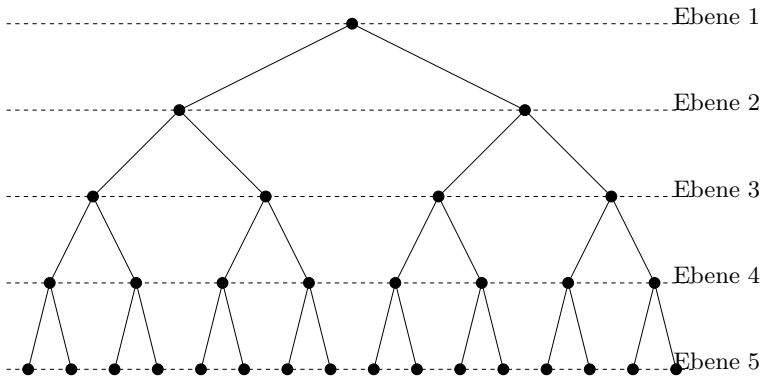


Abbildung 5: Ein Binärbaum zur Illustration

erzeugen, benutzen wir folgenden Algorithmus:

```
for k := 1 to n do
      1. in Ebene ℓ_k + 1 nimm Knoten
            v als c_k mit v ≠ c_i
            für alle i < k mit ℓ_i = ℓ_k
      2. entferne alle Nachfolger von v in T.
```

Da (wegen Punkt 2. des Schleifenrumpfes) stets nur Blätter in dem gleichzeitig konstruierten Baum T gewählt werden, erzeugt der Algorithmus einen Präfixcode. Die Frage ist nun, ob er für jedes $k = 1, \ldots, n$ in Punkt 1. des Schleifenrumpfes immer einen Knoten v in Ebene $\ell_k + 1$ findet. Um dies zu zeigen, betrachten wir die folgenden Ungleichungen:

$$2^m \geq 2^m \cdot \sum_{i=1}^{n} 2^{-\ell_i} > 2^m \cdot \sum_{i=1}^{k-1} 2^{-\ell_i} = \sum_{i=1}^{k-1} 2^{m-\ell_i}$$

Die rechte Größe beschreibt die Anzahl der Blätter des vollständigen Binärbaumes, die wir in den Schleifendurchläufen $1, 2, \ldots, k-1$ entfernt haben. Weil am Anfang T 2^m Blätter hat und weil $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_n$ gilt, folgt aus den obigen Ungleichungen, dass wir in Schritt k mindesten ein Blatt in Ebene $m + 1$ haben. Deshalb gibt es einen Knoten v wie in Punkt 1. des Schleifenrumpfes erforderlich. □

Aus dem Beweis des vorigen Satzes lesen wir ab:

Folgerung 2.2 *Jeder Präfixcode genügt der Kraftschen Ungleichung.*

Für das Alphabet $\Sigma = \{a_1, ..., a_k\}$, wobei jedes Symbol a_i mit Wahrscheinlichkeit $P(a_i)$ auftritt, und die Codierung $K : \Sigma \to \{0,1\}^+$ definieren wir die *erwartete Länge* von K als

$$L_K := \sum_{i=1}^{k} P(a_i) \cdot |K(a_i)|.$$

Unten erklären wir Shannon-, Shannon-Fano- und Huffman-Codierungen. Huffman-Codes sind optimale Präfixcodes (also Codes mit kleinster erwarteter Länge L_K) und die ersten beiden Codierungen erzeugen für manche Eingaben —wie wir sehen werden— nicht optimale Lösungen. Wir geben aber diese Verfahren an, weil sie interessante und nützliche Strategien für Datenkompression enthalten.

2.2 Shannon-Algorithmus

Es seien $p_1 = P(a_1)$, ..., $p_n = P(a_n)$ mit $p_1 \geq p_2 \geq \ldots \geq p_n$. Dann sei $P_1 := 0$, und für $i > 1$ sei $P_i := p_1 + \ldots + p_{i-1}$.

Shannon-Algorithmus:

```
for i := 1 to n do
        ℓ_i := ⌈− log p_i⌉;
        Sei  P_i := 0, b_1 b_2 b_3 ...
                K(a_i) := b_1 b_2 ... b_{ℓ_i};
```

Beispiel 2.3 *Betrachten wir einen Text über dem Alphabet $\Sigma = \{a, b, c, d, e\}$, wobei die Verteilung der Symbole durch $P(a) = 0,35$, $P(b) = 0,17$, $P(c) = 0,17$, $P(d) = 0,16$ und $P(e) = 0,15$ gegeben ist. Folgende Tabelle stellt den Code dar, wie er durch den Shannon-Algorithmus generiert wird:*

	p_i	ℓ_i	P_i	P_i *(Binär)*	*Code*
a	*0,35*	*2*	*0,0*	*0,0000000...*	*00*
b	*0,17*	*3*	*0,35*	*0,0101100...*	*010*
c	*0,17*	*3*	*0,52*	*0,1000010...*	*100*
d	*0,16*	*3*	*0,69*	*0,1011000...*	*101*
e	*0,15*	*3*	*0,85*	*0,1101100...*	*110*

Damit ergibt sich

$$L_S = 0,35 \cdot 2 + 0,17 \cdot 3 + 0,17 \cdot 3 + 0,16 \cdot 3 + 0,15 \cdot 3 = 2,65$$

Es ist leicht zu sehen, dass die Shannon-Codierung nicht optimal ist.

Die Shannon-Codierung zeigt in einfacher Weise Bezüge zwischen Zahldarstellungen von Wahrscheinlichkeiten (bzw. deren Summen) und Binärcodes.

Satz 2.4 *Der Shannon-Algorithmus generiert einen Präfixcode.*

Beweis: Es sei $P_i = 0, b_1 b_2 b_3 \ldots = \frac{b_1}{2^1} + \frac{b_2}{2^2} + \cdots$ Nach Konstruktion gilt

$$\log \frac{1}{p_i} \leq \ell_i,$$

womit für jedes $j \geq i+1$

$$P_j - P_i \geq P_{i+1} - P_i = p_i \geq \frac{1}{2^{\ell_i}}$$

gilt. Wegen $p_1 \geq p_2 \geq \ldots \geq p_n$ gilt $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_n$. Angenommen, es gibt ein i und ein j mit $i < j$ und

$$K(a_i) = b_1 b_2 \ldots b_{\ell_i}, \quad K(a_j) = c_1 c_2 \ldots c_{\ell_j}$$

wobei $\ell_j \geq \ell_i$. Ist $b_1 = c_1, \ldots, b_{\ell_i} = c_{\ell_i}$, so gilt

$$\begin{aligned} & P_j - P_i \\ = \; & \left(\frac{b_1}{2^1} + \cdots + \frac{b_{\ell_i}}{2^{\ell_i}} + \frac{c_{\ell_i+1}}{2^{\ell_i+1}} + \cdots\right) - \left(\frac{b_1}{2^1} + \cdots + \frac{b_{\ell_i}}{2^{\ell_i}} + \frac{b_{\ell_i+1}}{2^{\ell_i+1}} + \cdots\right) \\ < \; & 2^{-\ell_i}, \end{aligned}$$

also ein Widerspruch. □

2.3 Shannon-Fano-Codierung

Ein weiteres Beispiel für einen Präfixcode ist die *Shannon-Fano-Codierung*:

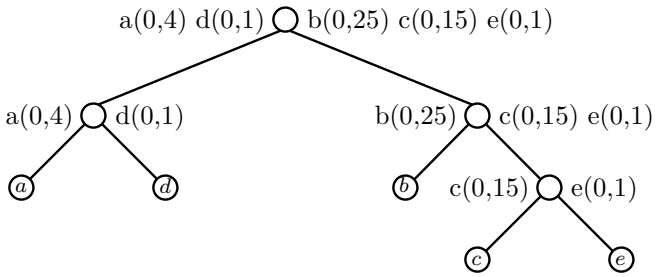
```
make-code(Σ)
    if |Σ| = 1 then for a ∈ Σ return node(a)
    else
        teile Σ auf in Σ1 und Σ2 mit ∑_{a∈Σ1} P(a) ≈ ∑_{a∈Σ2} P(a);
        return node(make-code(Σ1),make-code(Σ2)).
```

Aus dem rekursiven Aufbau des Algorithmus (Rekursionsbaum) folgt sofort:

Satz 2.5 *Der Shannon-Fano-Algorithmus liefert einen Präfixcode.*

Die Codierung zeigen wir nur an einem Beispiel. Im Vergleich mit der Huffman-Codierung ist die Methode von Shannon-Fano viel schlechter. Sie konstruiert nicht immer optimale Präfixcodes.

Beispiel 2.6 *Betrachten wir die folgende Eingabe:* $P(a) = 0,4$, $P(b) = 0,25$, $P(c) = 0,15$ *und* $P(d) = P(e) = 0,1$. *Der Shannon-Fano-Algorithmus arbeitet folgendermaßen*



Die Ausgabecode ist:

	P	Code	Länge
a	$0,4$	00	2
b	$0,25$	10	2
c	$0,15$	110	3
d	$0,1$	01	2
e	$0,1$	111	3

Für diese Codierung ist die erwartete Länge

$$L_{S\text{-}F} = 2 \cdot 0,4 + 2 \cdot 0,25 + 3 \cdot 0,15 + 2 \cdot 0,1 + 3 \cdot 0,1 = 2,25$$

während die (optimale) erwartete Länge $2,15$ *ist. Wir werden später sehen, daß die erwartete Länge für die Huffman-Codierung*

$$L_H = 1 \cdot 0,4 + 2 \cdot 0,25 + 3 \cdot 0,15 + 2 \cdot 4 \cdot 0,1 = 2,15$$

beträgt, also optimal ist.

Wie man leicht nachrechnet, liefert die Shannon-Fano-Codierung im Beispiel aus dem letzten Abschnitt eine erwartete Länge $L_{S-F} = 2,5$, ist also in dem Fall etwas besser als die Shannon-Codierung.

2.4 Huffman-Algorithmus

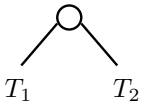
Huffman-Codes sind optimale Präfixcodes (also Codes mit kleinster erwarteter Länge L_K).

Der Algorithmus für die *Huffman-Codierung*:

1. Starte mit dem Wald aus Bäumen, in dem jeder Baum ein Symbol darstellt und $w_i = P(a_i)$ das Gewicht des Baumes ist.

2. Repeat until Wald besteht aus nur einem Baum:

   - wähle die zwei Bäume T_1 und T_2 mit den kleinsten Gewichten w_1 und w_2;
   - nimm nun statt T_1 und T_2 den Baum

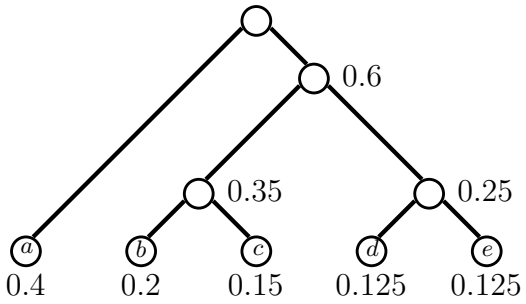


   mit dem Gewicht $w_1 + w_2$.

Beispiel 2.7 *Betrachten wir das Eingabealphabet* $\Sigma = \{a, b, c, d, e\}$ *und die Wahrscheinlichkeiten:*

$P(a)$	$P(b)$	$P(c)$	$P(d)$	$P(e)$
$0,4$	$0,2$	$0,15$	$0,125$	$0,125$

Der Algorithmus konstruiert den Ausgabebaum folgendermaßen:



Der Ausgabecode ist:

a	b	c	d	e
0	100	101	110	111

Man kann induktiv das folgende Ergebnis beweisen:

Satz 2.8 *Die erwartete Codelänge ist für Huffman-Codierung optimal.*

Betrachten wir jetzt die Qualität der Huffman-Codierung in Vergleich mit der Entropie, oder —anders gesagt— vergleichen wir nun die optimale erwartete Codelänge mit der Entropie.

Satz 2.9 *Sei Σ ein Quellenalphabet und $K : \Sigma \to \{0,1\}^+$ ein beliebiger Präfixcode. Dann gilt:*

$$L_K \geq H(\Sigma).$$

Satz 2.10 *Für jedes Quellenalphabet Σ ist die minimale erwartete Codelänge für Präfixcodes höchstens $H(\Sigma) + 1$.*

Dann bekommen wir als Korollar die folgende Abschätzung für die erwartete Länge der Huffman-Codierung:

Korollar 2.11 *Für jedes Quellenalphabet Σ gilt:*

$$H(\Sigma) \leq L_H \leq H(\Sigma) + 1.$$

Wir beschließen dieses Kapitel mit der Skizze einer Implementierung von Huffman Algorithmus.

Codierer:

```
/* --- first pass --- */
initialize_frequencies;
while (ch != eof)
{
     ch = getchar(input);
     update_frequence(ch);
}
construct_tree(T);
puttree(output,T);
/* --- second pass --- */
initialize_input;
while (ch != eof)
{
     ch = getchar(input);
     put(output,encode(ch));
}
put(output,encode(eof));
```

Decodierer:

```
gettree(input,T);
while ((ch=decode(input)) != eof)
     putchar(output,ch);
```

2.5 Erweiterte Huffman-Codierung

Betrachten wir motivierend folgendes Beispiel: Sei $\Sigma = \{a, b\}$ und $P(a) = 0,9$; $P(b) = 0,1$. Dann liefert der Huffman-Algorithmus folgende Codierung:

	P	Code
a	$0,9$	0
b	$0,1$	1

mit der erwarteten Länge $1 \cdot 0,9 + 1 \cdot 0,1 = 1$ Bits/Symbol. Die Entropie für unser Alphabet (also die mittlere Anzahl von Bits, um das Symbol aus Σ zu codieren) ist

$$H(\Sigma) = 0,9 \cdot \log \frac{10}{9} + 0,1 \cdot \log 10 = 0,47 \ .$$

Die *Redundanz* —die Differenz zwischen der erwarteten Länge und der Entropie— ist $0,53$ Bits je Symbol, also fast 113% der Entropie. Das heißt, daß (1) unser Code praktisch nicht komprimiert (er „übersetzt“ lediglich $a \to 0$ und $b \to 1$) und (2) um einen Text zu codieren, benutzen wir um 113% mehr Bits, als eine minimale Codierung bräuchte. Um diesen Unterschied zu verkleinern, betrachten wir das neue Alphabet

$$\Sigma^2 = \{aa, ab, ba, bb\}$$

mit der Wahrscheinlichkeiten: $P(aa) = [P(a)]^2$, $P(bb) = [P(b)]^2$ und $P(ab) = P(ba) = P(a) \cdot P(b)$. Der Huffman-Algorithmus konstruiert folgende Codierung:

	P	Code	Länge
aa	$0,81$	0	1
ab	$0,09$	10	2
ba	$0,09$	110	3
bb	$0,01$	111	3

Die erwartete Länge ist $1 \cdot 0,81 + 2 \cdot 0,09 + 3 \cdot 0,09 + 3 \cdot 0,01 = 1,29$ Bits je Symbol im Alphabet Σ^2, also $0,645$ Bits je Symbol in Σ und jetzt benutzt unsere Codierung nur noch 37% mehr Bits als eine minimale Codierung. Für das Alphabet

$$\Sigma^3 = \{aaa, aab, aba, baa, abb, bab, bba, bbb\}$$

ist der Unterschied noch kleiner:

	P	Code	Länge
aaa	$0,729$	0	1
aab	$0,081$	100	3
aba	$0,081$	101	3
baa	$0,081$	110	3
abb	$0,009$	11100	5
bab	$0,009$	11101	5
bba	$0,009$	11110	5
bbb	$0,001$	11111	5

und die erwartete Länge ist $0,52$ bps: nur um 11% mehr als die Entropie $H(\Sigma)$.

Sei H^m die *erweiterte Huffman-Codierung* für die Blöcke der Länge m. Dann bekommen wir die folgende Ungleichung.

Satz 2.12 *Für jedes Quellenalphabet Σ gilt:*

$$H(\Sigma) \leq L_{H^m} \leq H(\Sigma) + \frac{1}{m}.$$

Beweisidee: Wir zeigen, daß $H(\Sigma^m) = mH(\Sigma)$. Daraufhin verwenden wir Korollar 2.11. □

Wir haben gesehen, daß die Wahrscheinlichkeit für jedes neue „Zeichen" $a_{i_1}a_{i_2}\ldots a_{i_m}$ des Alphabets Σ^m das Produkt von Wahrscheinlichkeiten

$$P(a_1) \times P(a_2) \times \ldots P(a_m)$$

ist. Das heißt, daß die Wahrscheinlichkeiten für die Zeichen innerhalb eines Blocks unabhängig sind. In der Praxis haben wir aber sehr selten mit einer solchen Situation zu tun. Für die englische Sprache z. B. ist die Wahrscheinlichkeit, dass das Zeichen 'a' im U. S. Grundgesetz vorkommt, gleich $0,057$, aber die Wahrscheinlichkeit für die Reihenfolge von 10 'a's ist nicht $0,057^{10}$ sondern 0. In der Praxis muß man für jedes m eine neue Statistik für Σ^m konstruieren.

Das ist aufgrund des exponentiellen Wachstums kaum machbar. Einen praktikablen Ausweg beschreibt das nächste Kapitel.

Literatur

[1] K. Bosch. *Elementare Einführung in die angewandte Statistik,* Vieweg, 1987.

[2] D. Salomon. *Data Compression; The Complete Reference,* Springer, 1998.

[3] K. Sayood. *Introduction to Data Compression,* Morgan Kaufmann Publishers, Inc., 1996. Neue erweiterte Auflage: 2000.

[4] F. Topsøe. *Informationstheorie,* Teubner, 1974.