Skriptteil zur Vorlesung:
Proinformatik - Funktionale Programmierung

Dr. Marco Block-Berlitz

30.Juli 2009

Notation und Einführung

Der folgende Abschnitt gibt eine kurze Einführung in die Codierungstheorie. Dabei werden notwendige Begriffe und Definitionen vorgestellt.

Daten, die für eine Übertragung codiert werden, sollen anschließend wieder eindeutig decodiert werden können (siehe Abbildung 1).

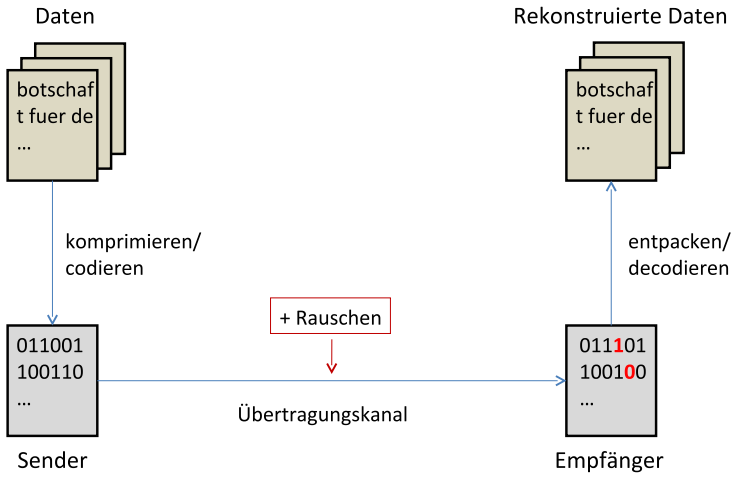

Abbildung 1: Übersicht

In einigen Fällen (z.B. Rauschen) kann es bei der Übertragung zu fehlerhaften Informationen kommen. Ziele der Codierungstheorie sind daher eine eindeutige Decodierbarkeit, kompakte (kurze) Codierungen und Fehler bei der Rekonstruktion nach Möglichkeit erkennen und korrigieren zu können.

Im Folgenden werden wir keine verlustbehafteten Codierungsverfahren (Komprimierung/Kompression) besprechen, wie die modernen und erfolgreichen Formate für Bilder, Musik oder Sprache (MPEG, JPEG, GIF, ...). An dieser Stelle sei auf die spezialisierte Literatur in [1, 5] verwiesen.

Alphabet und Wörter

Die endliche, nicht-leere Menge von Symbolen eines Datenraumes bezeichnen wir als **Alphabet** oder Σ. Damit läßt sich der Datenraum beschreiben, den es zu codieren gilt. So beispielsweise Symbole in Texten mit

$$\Sigma = \{a, b, ..., z, A, B, ..., Z, 0, 1, ..., 9\}.$$

Zum anderen kann ein weiteres Alphabet den Datenraum beschreiben, der uns für eine Übertragung zur Verfügung steht. In den meisten Fällen ist dieser Datenraum binär mit

$$\Sigma = \{0, 1\}.$$

Ein **Wort** (String) über Σ ist eine endliche, möglicherweise leere Folge von Symbolen aus dem Alphabet Σ. Die leere Folge (das leere Wort) bezeichnen wir mit ε.

Die **Länge** eines Wortes ist die Anzahl der vorkommenden Symbole, so gilt beispielsweise $|\varepsilon| = 0$ und $|x_1x_2...x_n| = n$.

Wir bezeichnen mit Σ^* die Menge aller Wörter und mit Σ^+ die Menge aller nicht-leeren Wörter über Σ, demzufolge ist

$$\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}.$$

Mit Σ^k bezeichnen wir die Menge aller Wörter über Σ mit der Länge k.

Damit läßt sich also die Menge alle Wörter Σ^* auch über die Wörter fester Längen Σ^k definieren

$$\Sigma^* = \bigcup_{k \geq 0} \Sigma^k$$

Um zwei Worte zu verbinden, verwenden wir die Konkatenation ($\circ$) mit

$$\_ \circ \_ : \Sigma^* \times \Sigma^* \to \Sigma^*.$$

Dadurch lassen sich erzeugte Wörter kombinieren, so gilt beispielsweise für das leere Wort $\varepsilon \circ w = w \circ \varepsilon = w$, für alle $w \in \Sigma^+$.

Codierung und Information

Der Begriff **Codierung** beschreibt im allgemeinen die Darstellung von Informationen durch Zeichenfolgen. Eine **Information** ist dabei ein Wort über dem Alphabet A.

Eine **codierte Information** ist ein Wort über dem Alphabet B. Dabei gilt meistens $B = \mathcal{B} = \{0, 1\}$ (Menge der Binärzeichen).

Für die Überführung einer Information über dem Alphabet A zum Alphabet B bedarf es einer injektiven Abbildung (siehe Abbildung 2) c mit

$$c : A \to B^+,$$

die fortgesetzt wird mittels $c(a_1a_2...a_k) = c(a_1) \circ c(a_2) \circ ... \circ c(a_k)$.

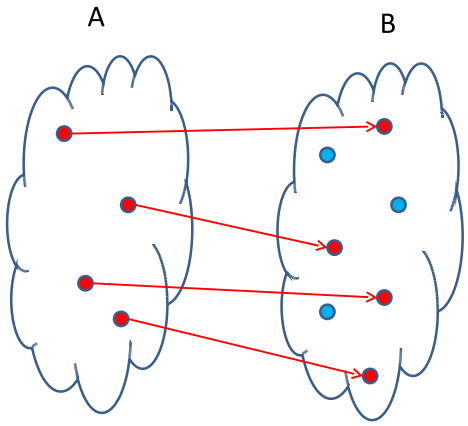

Abbildung 2: Injektive Abbildung von der Menge A zur Menge B. Jeder Punkt in A wird höchstens einmal in der Zielmenge B getroffen. Das ist notwendig, damit jedes Element aus A eine Codierung erhält, die später im Rekonstruktionsschritt eindeutig zum Element zurückführt.

Die **Menge der Codewörter** für ein Alphabet A (nennen wir auch kurz Code) ist $C(A) = \{c(a)|a \in A\}$. So könnte beispielsweise für $A = \{a, b, c\}$ die Menge der Codewörter die folgende sein $C(A) = \{00, 01, 10\}$.

Um einen eindeutig decodierbaren Code zu erhalten muss eine Umkehrfunktion d existieren, mit

$$d : \{c(w)|w \in A^+\} \to A^+.$$

Dieser muss eindeutig decodierbar sein, also muss gelten $d(c(w)) = w$.

Codierung mit fester Codewortlänge

Die einfachste Möglichkeit einen Code zu erstellen, ergibt sich durch den sogenannten Blockcode, bei dem die Länge aller Codewörter für die Einzelzeichen gleich ist. Im nachfolgenden Kapitel werden wir dann auch Codes mit variablen Codewortlängen kennenlernen.

Blockcode

Im Blockcode c wird jedes Zeichen durch ein $0-1-$Wort der gleichen Länge n codiert mit $c : A \to \mathcal{B}^n$. Mit einem Blockcode der Länge n lassen sich maximal 2^n Zeichen codieren.

Dazu identifizieren wir die i-te Zweierpotenz, mit der alle Symbole codiert werden können, mit

$$2^{i-1} < |A| \leq 2^i.$$

So benötigen wir beispielsweise für die Symbolmenge $A = \{a, b, c\}$ zwei Bit, denn $|A| = 3 \leq 2^2$. Die Codierung könnte dann wie folgt aussehen:

$c(s)$	s
00	a
01	b
10	c
11	–

Da alle Codewörter gleich lang sind, können wir über Abstandsfunktionen hilfreiche Eigenschaften von speziellen Codierungen beschreiben. Diese können uns, so werden wir es später sehen, auch bei der Rekonstruktion von Fehlern helfen. Als wichtigste Abstandsfunktion verwenden wir dabei den Hamming-Abstand.

Hamming-Abstand

Seien $b, b' \in \mathcal{B}^n$ mit $b = (b_1, ..., b_n)$ und $b' = (b'_1, ..., b'_n)$, dann sei der Hamming-Abstand (siehe Abbildung 3) wie folgt definiert

$$d_H(b, b') = |\{i | b_i \neq b'_i\}|.$$

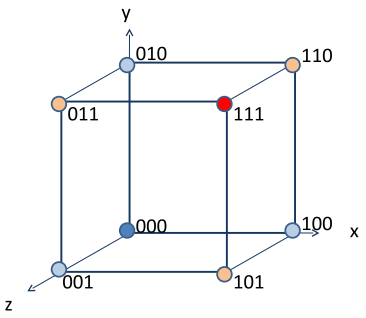

Abbildung 3: Hamming-Abstand: In diesem Beispiel gibt es die Codewörter 000 (blau) und 111 (rot). Der Abstand $d_H(000, 111) = 3$ ist so zu interpretieren, dass wir mindestens 3 Kanten entlang laufen müssen, um vom Codewort 000 zu 111 zu gelangen.

Für einen Blockcode c sei der Abstand gerade definiert als das Minimum aller Abstände

$$d_H(c) = min\{d_H(c(a), c(a')) | a \neq a' \wedge a, a' \in A\}.$$

Der Hammingabstand dient im Folgenden als Kenngröße zur Charakterisierung der Fähigkeit einer Blockcodes zu Fehlererkennung und -korrektur.

Angenommen, wir haben die folgenden Codewörter $C = \{001, 110, 111\}$ über einen Blockcode c erhalten, dann sind die paarweisen Abstände

$$d_H(001, 110) = 3$$

$$d_H(001, 111) = 2$$

$$d_H(110, 111) = 1.$$

Das Minimum aller Abstände $min\{3, 2, 1\} = 1$, demzufolge ist $d_H(c) = 1$.

Eigenschaften von Metriken

Es lassen sich aber auch andere Metriken (Abstandsfunktionen) d verwenden, mit $d : X \times X \to \mathbb{R}$ ist Metrik für Menge X, falls für beliebige $x, y, z \in X$ die folgenden Axiome erfüllt sind

(1) $d(x, y) \geq 0$, $\forall x, y \in X$

(2) $d(x, y) = 0 \iff x = y$ (Definitheit)

(3) $d(x, y) = d(y, x)$, $\forall x, y$ (Symmetrie)

(4) $d(x, z) \leq d(x, y) + d(y, z)$, $\forall x, y, z \in X$ (Dreiecksungleichung)

Zwei weitere wichtige Metriken sind beispielsweise für $r = (x, y), r' = (x', y') \in \mathbb{R}^2$

- Manhattan-Abstand $d_1$mit $d_1(r, r') = |x - x'| + |y - y'|$ und
- Euklidischer Abstand d_2 mit $d_2(r, r') = \sqrt{(x - x')^2 + (y - y')^2}$.

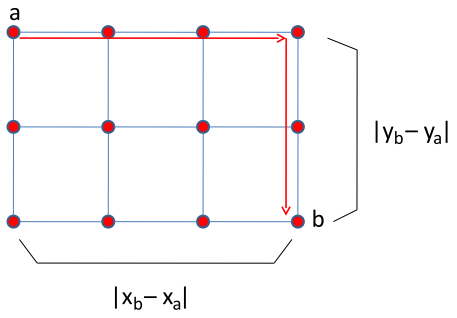

Abbildung 4: Beim Manhattan-Abstand werden die Differenzen komponentenweise berechnet und aufsummiert. Die Bezeichnung bezieht sich auf die Straßenverläufe Manhattans.

Die Decodierung eines Blockcodes mittels Maximum-Likelihood-Methode: Eine Nachricht $b \in \mathcal{B}^n$ wird decodiert als Zeichen $a \in A$ mit $d_H(c(a), b)$ ist minimal für alle a.

Fehlererkennender Code

Sei $c : A \to \mathcal{B}^n$ ein Blockcode, dann heißt c k-fehlererkennend, wenn $\forall a \in A$, $b \in \mathcal{B}^n$ gilt, dass

$$1 \leq d_H(b, c(a)) \leq k \Rightarrow b \notin C(A).$$

D.h. wenn sich ein empfangenes Wort in bis zu k Stellen von einem Codewort unterscheidet, dann ist es selbst kein Codewort.

Eine graphische Interpretation, die die Eigenschaft k-fehlererkennend illustriert, ist Abbildung 5 zu entnehmen.

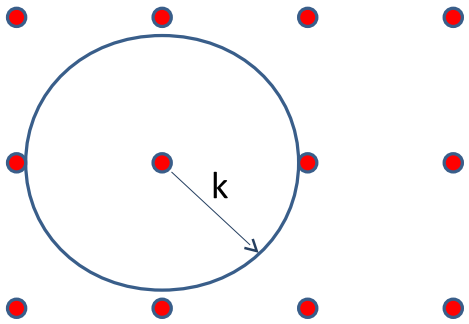

Abbildung 5: In dieser zweidimensionalen Abbildung läßt sich die Eigenschaft k-fehlererkennend anschaulich interpretieren. So läßt ein Abstand k von jedem Datenpunkt aus $\mathcal{B}^n$ mindestens k Fehler erkennen. Eine erfolgreiche Rekonstruktion der Daten spielt dabei keine Rolle.

Eine Codierung C ist allgemein k-fehlererkennend, wenn $d_H(C) \geq k + 1$.

Wenn wir beispielsweise die Codierungen aus Abbildung 3 verwenden, dann hat dieser folglicherweise die Eigenschaft 2-fehlererkennend zu sein. Bei einem Fehler von 000 aus landen wir in den hellblauen und bei zwei Fehlern in den orangen Bereichen. Da also der minimale Abstand von allen Codewörtern zueinander drei ist, können zwei Fehler erkannt werden.

Fehlerkorrigierender Code

Sei $c : A \to \mathcal{B}^n$ ein Blockcode, dann heißt c k-fehlerkorrigierend, wenn $\forall a \in A$, $b \in \mathcal{B}^n$ gilt, dass

$$d_H(b, c(a)) \leq k \Rightarrow \forall a' \neq a : d_H(b, c(a')) > k \ .$$

Die Abbildung 6 liefert eine graphische Interpretation.

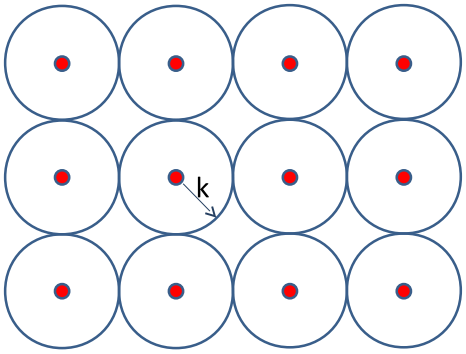

Abbildung 6: Bei der Eigenschaft k-fehlerkorrigierend gibt es für jeden Datenpunkt aus $\mathcal{B}^n$ ein k für das eine erfolgreiche Rekonstruktion vorgenommen werden kann.

Wenn c k-fehlerkorrigierend und b eine fehlerhafte Nachricht ist, kann $c(a)$ als eindeutiges Codewort (bei Maximum-Likelihood-Decodieren siehe Abschnitt ??) mit $\leq k$ Fehlern zugeordnet werden.

Als Beispiel soll wieder die Abbildung 3 dienen. Es ist leicht zu sehen, dass der Code 1-fehlerkorrigierend ist. Eine Codierung C ist allgemein k-fehlerkorrigierend, wenn $d_H(C) \geq 2k + 1$.

Informationsrate

Um den Informationsgehalt für eine Codierung zu ermitteln, lässt sich die **Informationsrate** IR mit

$$IR = \frac{\#Informationsbit}{Codewortlaenge}$$

ermitteln. Zu den im folgenden Abschnitt vorgestellten Blockcode-Varianten werden wir die dazugehörigen Informationsraten ermitteln.

Blockcode-Varianten

Jetzt gibt es viele verschiedene Möglichkeiten Blockcodes zu erstellen, die sich in Informationsrate, Fehlererkennung und -korrektur unterscheiden. Im Folgenden werden ein paar elementare Varianten vorgestellt.

Variante 1: Paritätsbit

Ein Blockcode c_p mit einem zusätzlichen Paritätsbit, mit $c_p : A \rightarrow \mathcal{B}^{n+1}$ codiert zunächst das Wort $c(a)$ und hängt anschließend das Paritätsbit $b_p(a)$, mit

$$b_p(c(a)) = \begin{cases} 0 & , \textit{gerade Anzahl von Einsen} \\ 1 & , \textit{sonst} \end{cases}$$

an, so dass per Definition

$$c_p(a) =_{def} \; c(a) \circ b_p(c(a)).$$

Daraus folgt, dass $d_H(c_p) \geq 2$, weil alle Codewörter $c_p(a)$ gerade Anzahl von Einsen haben. Die Informationsrate beträgt $IR(c_p) = \frac{n}{n+1}$.

Soll beispielsweise das Codewort $c(s) = 110$ mit Paritätsbit angegeben werden, so ist $c_p(s) = 1100$.

Variante 2: Doppelcodierung

Bei der Doppelcodierung c^2 schreiben wir den Code einfach zweimal hin $c^2 : A \to \mathcal{B}^{2n}$, mit

$$c^2(a) =_{def} \; c(a) \circ c(a).$$

Daraus läßt sich ableiten, dass $d_H(c^2) \geq 2$. Fehlerkorrigierend ist diese Codierung aber nicht zwangsläufig. Die Informationsrate beträgt $IR(c^2) = \frac{n}{2n} = \frac{1}{2}$.

Beispielsweise ist für $c(s) = 0110$ der Code mit Doppelcodierung $c^2(s) = 01100110$.

Variante 3: Doppelcodierung mit Paritätsbit

Verwenden wir zusätzlich zur Doppelcodierung noch ein Paritätsbit $c_p^2 : A \to \mathcal{B}^{2n+1}$, mit

$$c_p^2(a) =_{def} \; c(a) \circ c(a) \circ b_p(c(a))$$

so läßt sich beobachten, dass $d_H(c_p^2) \geq 3$. Damit ist c_p^2 sogar 1-fehlerkorrigierend.

Beweis: $d_H(c(a), c(a')) \geq 2 \Rightarrow d_H(c_p^2(a), c_p^2(a')) \geq 4$. Nun ist aber der Fall interessant, bei dem $d_H(c(a), c(a')) = 1$, dann ist aber $d_H(c_p^2(a), c_p^2(a')) = 3$, da der Abstand verdoppelt wird und das Paritätsbit kippt.

Für die Doppelcodierung mit Paritätsbit erhalten wir folgende Informationsrate

$$IR(c_p^2) = \frac{n}{2n+1} < \frac{1}{2}.$$

Aufpassen muss man lediglich beim Paritätsbit, da für $c(s) = 1011$ das letzte Bit für $c_p^2(s) = 101110111$ nur aus dem Codewort $c(s)$ ergibt.

Übersicht der bisherigen Codierungen

Wir erkaufen uns die Möglichkeit zur Fehlererkennung bzw. -korrektur durch zusätzliche Bits, die für den Informationsgehalt redundant sind.

z	$c(z)$	$c_p(z)$	$c^2(z)$	$c_p^2(z)$
a	00	000	0000	00000
b	01	011	0101	01011
c	10	101	1010	10101
d	11	110	1111	11110

Es ist leicht zu sehen, dass $c^2(z)$ aus Spalte 4 trotz doppelter Datenmenge nicht 1-fehlerkorrigierend ist, da beispielsweise a mit einem Fehlbit an der ersten Position den gleichen Hamming-Abstand hätte, wie c an der dritten Position:

$$d_H(c^2(a), 1000) = d_H(c^2(c), 1000) = 1.$$

Die Spalte 5 hingegen ist 1-fehlerkorrigierend.

Variante 4: Kreuzsicherungscode

Beim Kreuzsicherungscode c_x werden die Codes in einer Tabelle angeordnet und für jede Zeile und jede Spalte jeweils ein Paritätsbit ermittelt (siehe Abbildung 7).

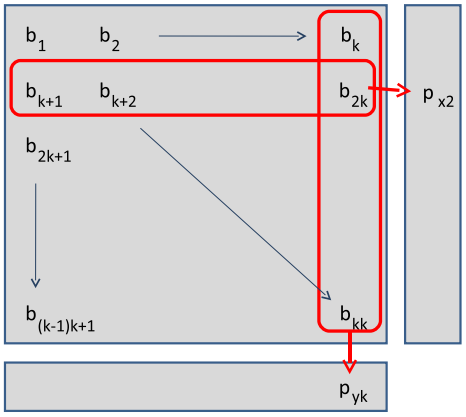

Abbildung 7: Der Kreuzsicherungscode bringt eine Nachricht in die Matrixform $n \times n$ und speichert zusätzlich für alle Zeilen und Spalten $2n$ Paritätsbits.

Variante 5: Graycode

Schauen wir uns einmal Tabelle 1 an. Dabei sehen wir, dass der Hammingastand zwischen zwei aufeinander folgende Binärzahlen groß ein kann, auch wenn die Dezimalzahlen Nachbarn sind.

Gerade bei genetischen Algorithmen, möchte man aber eine lokale Nähe in der binären Repräsentation haben, wenn diese in der Orginalproblembeschreibung ebenfalls nahe beieinander liegen. Daher wird dort oft, anstatt der normalen Binärdarstellung, die Graycode-Darstellung gewählt.

Binär- nach Graycode

Wir erzeugen aus einer gegebenen Binärdarstellung $b_1 b_2 ... b_n$ elementeweise die Einträge für g mit

$$\gamma : g_i = \begin{cases} b_i & , falls\, i = 1 \\ b_{i-1} \oplus b_i & , falls\, i > 1 \end{cases}.$$

Int	Binärcode	Hammingabstand	Graycode	Hammingabstand
0	000	-	000	-
1	001	1	001	1
2	010	2	011	1
3	011	1	010	1
4	100	3	110	1
5	101	1	111	1
6	110	2	101	1
7	111	1	100	1

Tabelle 1: Vergleich der Hammingabstände zwischen normaler Binär- und Graycodierung.

Dabei ist ($\oplus$) gerade die Addition modulo 2 oder einfach die XOR-Funktion.

Gray- nach Binärcode

Die Umkehrfunktion berechnet sich dabei wie folgt

$$\gamma^{-1} : b_i = \oplus_{j=1}^{i} g_j.$$

Da es sich bei dem vorliegenden Skript um eine Einführung in die Codierungstheorie handelt, können die Codierungen mit festen Codewortlängen nur kurz angerissen werden. Als weiterführende Literatur empfehle ich an dieser Stelle die Bücher von Wilfried Dankmeier [1] und Ralph-Hardo Schulz [2].

Literatur

[1] Dankmeier D.: „*Grundkurs Codierung: Verschlüsselung, Kompression, Fehlerbeseitigung.*“, 3.Auflage, Vieweg-Verlag, 2006

[2] Schulz R.-H.: „*Codierungstheorie: Eine Einführung*“, 2.Auflage, Vieweg+Teubner, 2003

[3] Schöning U.: „*Algorithmik*“, ISBN-13: 978-3827410924, Spektrum Akademischer Verlag, 2001

[4] Cormen T.H., Leiserson C.E., Rivest R.L.: „*Introduction to Algorithms*“, MIT-Press, 2000

[5] Wikibook zur Datenkompression: *http://de.wikibooks.org/wiki/Datenkompression*

[6] Haskell-Funktionen: *http://www.zvon.org/other/haskell/Outputglobal/index.html*