1. Einführung und wichtigste Definitionen

Codierungstheorie
Anton Malevich
14.02.2018

Definitionen

Code

Ein *Code* der Länge n über dem Alphabet K ist eine Menge $C \subset K^n$. Ein Element $v \in C$ heißt ein *Codewort*. Elemente aus K^n heißen generell *Worte*.

Hamming-Abstand

Hamming-Abstand $d(v, w)$ von zwei Worten $v = (v_1, \ldots, v_n)$, $w = (w_1, \ldots, w_n) \in K^n$ ist die Anzahl der Stellen, bei denen sich v und w unterscheiden:

$$d(v, w) = \#\{i : v_i \neq w_i\}.$$

```
HammingDistance[{1, 0, 1, 0, 1}, {1, 1, 1, 1, 1}]
```

2

Eine *(Hamming-)Kugel* $B_r(v)$ mit Mittelpunkt v und Radius $r \geqslant 0$ ist die Menge aller Worten in K^n mit Abstand höchstens r zu v:

$$B_r(v) = \{w \in K^n : d(v, w) \leqslant r\}.$$

Zum Beispiel, für $K = \{0, 1\}$ ist

$$B_1(\{1, 0, 1\}) = \{\{1, 0, 1\}, \{0, 0, 1\}, \{1, 1, 1\}, \{1, 0, 0\}\}.$$

Ein Code C heißt *perfekt*, falls es ein $r \geqslant 0$ gibt, sodass K^n eine disjunkte Vereinigung der Kugel von Radius r mit Mittelpunkten in C wird:

$$K^n = \bigcup_{c \in C} B_r(c), \text{ wobei } B_r(c) \bigcap B_r(c') = \emptyset \text{ für } c \neq c'.$$

Minimaldistanz

Minimaldistanz $d(C)$ eines Codes $C \subset K^n$ ist der kleinste Abstand zwischen zwei verschidenen Codeworten im Code C:

$$d(C) = \min\{d(v, w) : v, w \in C,\ v \neq w\}.$$

Der Code C heißt *t-fehlererkennend*, falls $d(C) \geqslant t + 1$, und *t-fehlerkorrigierend*, falls $d(C) \geqslant 2t + 1$ ist.

Maximum-Likelihood-Decodierung

Angenommen, es ist ein Codewort $c \in C$ versendet worden, und ein Wort $y \in K^n$ empfangen worden. Dann wird y zu dem Codewort $c' \in C$ decodiert, der den kleinsten Abstand zu y hat:

$y \to c'$ mit $d(y, c') = \min\{d(y, v) : v \in C\}$.

Aufgaben

Aufgabe 1

- Gegeben sei der binäre Code C, der aus den folgenden Codeworten besteht

$$C = \begin{pmatrix} 1&0&0&0&0&0&1\\ 1&0&0&1&1&1&1\\ 0&1&1&0&0&0&0\\ 1&0&1&0&1&1&0\\ 1&1&0&0&1&0&0\\ 0&1&1&1&1&1&0\\ 1&0&1&1&0&0&0\\ 1&1&0&1&0&1&0\\ 0&1&0&0&1&1&1\\ 0&0&1&0&1&0&1\\ 1&1&1&0&0&1&1\\ 0&1&0&1&0&0&1\\ 0&0&1&1&0&1&1\\ 1&1&1&1&1&0&1\\ 0&0&0&0&0&1&0\\ 0&0&0&1&1&0&0 \end{pmatrix};$$

  Bestimmen Sie die Minimaldistanz von C und zeigen Sie, dass C 1-fehlerkorrigierend ist.

  Decodieren Sie die Worten $y_1 = 1\,001\,001$ und $y_2 = 1\,001\,100$.

- Beweisen Sie, dass der Code C perfekt ist.

Aufgabe 2

Vorbereitung

In dieser Aufgabe wollen wir Nachrichten codieren, übersenden und decodieren. Die Nachricht wird aus lateinischen Buchstaben und weiteren Symbole bestehen; zur Konvertation in die binäre Form benutzen wir in Mathematica eingebaute Funktionalität.

```
ToCharacterCode["0,p"]
```

```
{48, 44, 112}
```

Für einen Buchstaben benutzen wir 1 Byte = 8 Bit. Zum Beispiel ist für "0":

```
IntegerDigits[48, 2, 8]
```

```
{0, 0, 1, 1, 0, 0, 0, 0}
```

Wir wollen den Code C aus Aufgabe 1 benutzen, der aber nur 16 Codeworte hat. Wir können also nur 4 Bit übersenden ($2^4 = 16$). Daher werden wir die 8-Bit lange Tupeln in zwei 4-Bit langen teilen, und diese versenden.

```
Partition[%, 4]
```

```
{{0, 0, 1, 1}, {0, 0, 0, 0}}
```

Nun brauchen wir eine Übersetzungstabelle, die wir zum Codieren und Decodieren benutzen werden.

```
CodeTable = MapThread[#1 ↔ #2 &, {Tuples[{0, 1}, 4], C}];
TableForm@%
```

```
{0, 0, 0, 0} ↔ {1, 0, 0, 0, 0, 0, 1}
{0, 0, 0, 1} ↔ {1, 0, 0, 1, 1, 1, 1}
{0, 0, 1, 0} ↔ {0, 1, 1, 0, 0, 0, 0}
{0, 0, 1, 1} ↔ {1, 0, 1, 0, 1, 1, 0}
{0, 1, 0, 0} ↔ {1, 1, 0, 0, 1, 0, 0}
{0, 1, 0, 1} ↔ {0, 1, 1, 1, 1, 1, 0}
{0, 1, 1, 0} ↔ {1, 0, 1, 1, 0, 0, 0}
{0, 1, 1, 1} ↔ {1, 1, 0, 1, 0, 1, 0}
{1, 0, 0, 0} ↔ {0, 1, 0, 0, 1, 1, 1}
{1, 0, 0, 1} ↔ {0, 0, 1, 0, 1, 0, 1}
{1, 0, 1, 0} ↔ {1, 1, 1, 0, 0, 1, 1}
{1, 0, 1, 1} ↔ {0, 1, 0, 1, 0, 0, 1}
{1, 1, 0, 0} ↔ {0, 0, 1, 1, 0, 1, 1}
{1, 1, 0, 1} ↔ {1, 1, 1, 1, 1, 0, 1}
{1, 1, 1, 0} ↔ {0, 0, 0, 0, 0, 1, 0}
{1, 1, 1, 1} ↔ {0, 0, 0, 1, 1, 0, 0}
```

So wird, zum Beispiel, die Nachricht "0,p" mit 6 Codeworten codiert:

```
ToCharacterCode["0,p"];
Partition[IntegerDigits[#, 2, 8], 4] & /@ %;
Flatten[%, 1];
messages = CodeTable⟦FromDigits[#, 2] + 1, 2⟧ & /@ %
```

```
{{1, 0, 1, 0, 1, 1, 0}, {1, 0, 0, 0, 0, 0, 1}, {0, 1, 1, 0, 0, 0, 0},
 {0, 0, 1, 1, 0, 1, 1}, {1, 1, 0, 1, 0, 1, 0}, {1, 0, 0, 0, 0, 0, 1}}
```

Diese codierten Nachrichten kann man nun versenden. Damit es spannend wird, simulieren wir einen fehlrerhaften Kanal, der mit kleiner Wahrscheinlichkeit *p* jedes Symbol *x* durch ein anderes aus $x \neq y \in K$ ersetzt. (*p* heißt *Symbolfehlerwahrscheinlichkeit.*)

```
p = 0.07;
Channel[x_, p_ /; RandomReal[] < p] := Mod[x + 1, 2]
Channel[x_, _] := x
SetAttributes[Channel, Listable]
```

```
Plus @@ Table[Channel[0, p], {1000}]
```

```
71
```

Nun “versenden” wir die codierten Nachrichten über unseren Kanal.

```
recieved = Channel[messages, p]
messages == recieved
```

```
{{1, 0, 1, 0, 1, 1, 1}, {1, 0, 0, 0, 0, 0, 1}, {0, 1, 1, 0, 0, 0, 0},
 {0, 0, 1, 1, 0, 1, 1}, {1, 1, 0, 1, 1, 1, 0}, {0, 0, 1, 0, 0, 0, 1}}
```

```
False
```

Wir decodieren nun die empfangenen Nachrichten. Dafür finden wir zuerst zu jedem empfangenen Wort jeweils das nächstgelegene Codewort. Der Fehler wird erkannt, falls das empfangene Wort kein Codewort ist. Der Fehler wird korrigiert, falls das nächstgelegene Codewort eindeutig ist. (In einem perfekten Code ist es immer der Fall, im Allgemeinen aber nicht! Es können zum Beispiel zwei Codeworten den gleichen Abstand zum empfangenen Wort haben.)

```
Outer[HammingDistance, recieved, C, 1, 1];
MatrixForm@%
decoded = Extract[C, First@Position[#, Min@#]] & /@ %%
```

$$\begin{pmatrix} 3 & 2 & 5 & 1 & 4 & 4 & 4 & 5 & 3 & 2 & 2 & 6 & 3 & 3 & 4 & 5 \\ 0 & 3 & 4 & 4 & 3 & 7 & 3 & 4 & 4 & 3 & 3 & 3 & 4 & 4 & 3 & 4 \\ 4 & 7 & 0 & 4 & 3 & 3 & 3 & 4 & 4 & 3 & 3 & 3 & 4 & 4 & 3 & 4 \\ 4 & 3 & 4 & 4 & 7 & 3 & 3 & 4 & 4 & 3 & 3 & 3 & 0 & 4 & 3 & 4 \\ 5 & 2 & 5 & 3 & 2 & 2 & 4 & 1 & 3 & 6 & 4 & 4 & 5 & 3 & 4 & 3 \\ 2 & 5 & 2 & 4 & 5 & 5 & 3 & 6 & 4 & 1 & 3 & 3 & 2 & 4 & 3 & 4 \end{pmatrix}$$

```
{{1, 0, 1, 0, 1, 1, 0}, {1, 0, 0, 0, 0, 0, 1}, {0, 1, 1, 0, 0, 0, 0},
 {0, 0, 1, 1, 0, 1, 1}, {1, 1, 0, 1, 0, 1, 0}, {0, 0, 1, 0, 1, 0, 1}}
```

Wir konvertieren dann mit der Tabelle, fügen je zwei 4-Bit Tupel zu einem 8-Bit Tupel zusammen und lesen das Symbol ab.

```
IntegerDigits[Position[CodeTable, #][[1, 1]] - 1, 2, 4] & /@ decoded
Join @@@ Partition[%, 2]
FromDigits[#, 2] & /@ %
FromCharacterCode /@ % // Row
```

```
{{0, 0, 1, 1}, {0, 0, 0, 0}, {0, 0, 1, 0}, {1, 1, 0, 0}, {0, 1, 1, 1}, {1, 0, 0, 1}}
```

```
{{0, 0, 1, 1, 0, 0, 0, 0}, {0, 0, 1, 0, 1, 1, 0, 0}, {0, 1, 1, 1, 1, 0, 0, 1}}
```

```
{48, 44, 121}
```

```
0,y
```

Aufgabe/Fragen

Versuchen Sie zu verstehen, was in jedem Schritt passiert.

Warum hat der Decodierer den Fehler gemacht und die Originalnachricht "0,p" als "0,y" decodiert?

Schreiben Sie eine Funktion für das Codieren von Textnachrichten in Codeworte und das Decodieren von empfangenen Worten in Textnachrichten.

Codieren Sie die Textnachricht “Coding is cool! :)”. Übersenden Sie die Codeworte über ein Kanal mit Symbolfehlerwahrscheinlichkeit $p = 0.01$, ein mit $p = 0.1$ und ein mit $p = 0.2$. Decodieren Sie jeweils die empfangenen Worte. Welche Schlussfolgerung kann man machen?

- Betrachten Sie nun C als einen ternären Code, d.h. das Alphabet ist nun $K = \{0, 1, 2\}$. Wie ändern sich das Codieren und das Decodieren. Der fehlerhafte Kanal mit Symbolfehlerwahrscheinlichkeit wird nun wie folgt simuliert.

```
Channel3[x_, p_ /; RandomReal[] < p] := If[RandomReal[] < .5, Mod[x + 1, 3], Mod[x + 2, 3]]
Channel3[x_, _] := x
SetAttributes[Channel3, Listable]
```

```
Table[Channel3[0, .1], {1000}];
Count[%, #] & /@ {0, 1, 2}
```

```
{899, 52, 49}
```