Datenkompression

Arithmetische Codierung

Digitale AV Technik, MIB 5

Aus Sicht der Informationstheorie

Problem	**Kompression**	**Fehlerkorrektur**
Ziel	Effizienz	Verlässlichkeit
Anwendung	**Quellencodierung**	Kanalcodierung

Algorithmische Perspektive

Problem	**Kompression**
Algorithmen	Shannon–Fano coding
	Huffman-Code
	Arithmetische Codierung, CABAC
	Lempel-Ziv-Welch

Arithmetische Codierung - Grundprinzip

Huffman Codes sind immer ganzzahlige Codes und daher immer bis zu 1 Bit schlechter als durch die Entropie eigentlich möglich.

Arithmetische Codierung umgeht das Problem indem nicht einzelne Zeichen jeweils einen Code zugeordnet bekommen, sondern ganze Nachrichten.

Beim arithmetischen Codieren wird einer Nachricht eine einzelne Bruchzahl zwischen 0 und 1 zugeordnet. Diese Zahl codiert die ganze Nachricht und ermöglicht auch die Dekodierung.

Arithmetische Codierung - Quelle

Die folgende Beschreibung ist aus einem Blogartikel von Ahmed Gad (Neptune.ai) übernommen.

Arithmetische Codierung - Eingabe

Um die Wahrscheinlichkeit jedes Symbols zu berechnen, benötigt der Algorithmus eine Häufigkeitstabelle als Eingabe. Diese Tabelle ordnet jedem Symbol seine Häufigkeit zu.

Zum Beispiel:

$a = 2$

$b = 7$

$c = 1$

Anhand der Häufigkeitstabelle wird eine Wahrscheinlichkeitstabelle erstellt, die die relativen Häufigkeiten der Symbole enthält.

Zum Beispiel:

$$p(a) = 2/10 = 0.2$$

$$p(b) = 7/10 = 0.7$$

$$p(c) = 1/10 = 0.1$$

Arithmetische Codierung - Kodierung

Bei der Kodierung werden die kumulativen Wahrscheinlichkeiten aller Symbole auf einer Linie dargestellt, die von 0.0 bis 1.0 reicht. Auf dieser Linie verwendet jedes Symbol einen Teilbereich. Man beschreibt diesen Bereich auch als Intervall $[0.0, 1.0)$, wobei es die 0.0 mit einschließt, die 1.0 aber nicht. Es gibt also nur Kodierungen die mit $0.$ beginnen.

Ein beliebiges Symbol C beginnt mit dem Wert S_l und endet mit dem Wert S_u, der nach der folgenden Gleichung berechnet wird:

$$S_u = S_l + (P(C)) * R$$

Wobei:

S: Die kumulative Summe aller vorherigen Wahrscheinlichkeiten.
$P(C)$: Die Wahrscheinlichkeit des Symbols C.
R: Der Bereich der Linie, der berechnet wird, indem man den Anfang vom Ende der Linie subtrahiert.

Am Anfang geht die Linie von 0.0 bis 1.0, also gilt $R = 1.0$.

Für die Symbole **a**, **b**, und **c** ergibt sich also:

Das erste Symbol beginnt am gleichen Startpunkt der Linie (0.0) und das letzte Symbol endet am Endpunkt der Linie (1.0).

Ein Symbol C deckt einen Prozentsatz des Bereichs ab, der seiner Wahrscheinlichkeit entspricht. Zum Beispiel deckt das Symbol b 70% der Linie ab, da seine Wahrscheinlichkeit 0.7 ist.

Für jedes Symbol in der Nachricht unterteilen wir die Bereiche immer weiter.

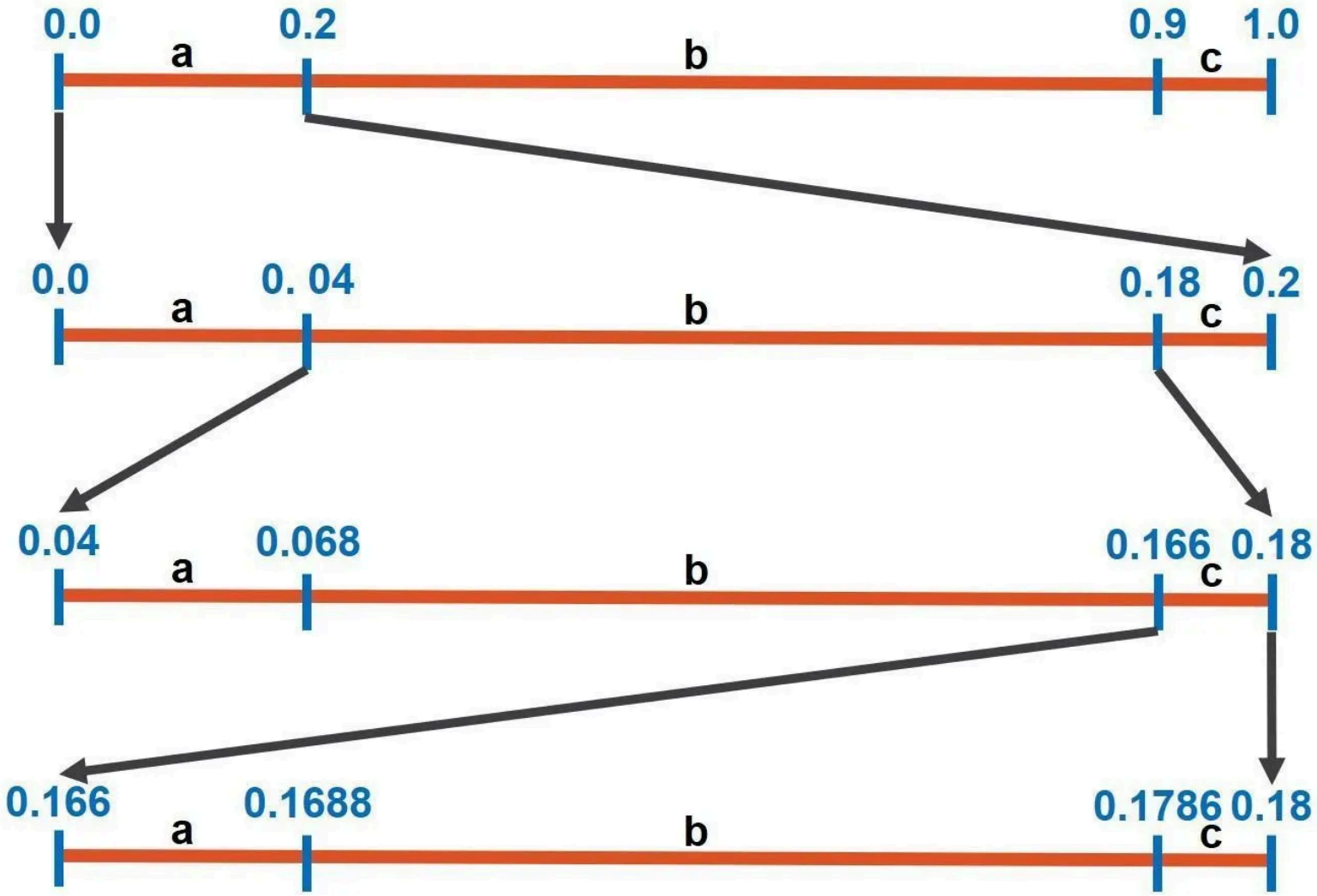
Um die Nachricht **abc** zu codieren, benötigen wir also eine Zahl im Bereich $[0.0, 0.2)$, da **a** das erste Symbol der Nachricht ist.

Wir unterteilen diesen Bereich dann wieder anhand der Wahrscheinlichkeiten unserer Symbole.

Wie genau die Unterteilung aussieht berechnen wir wieder mit der selben Formel:

$$S_u = S_l + (P(C)) * R$$

Wir wiederholen dies bis alle Symbole der Nachricht kodiert sind:



Das letzte Intervall reicht von 0.166 bis 0.18. Innerhalb dieses Bereichs kann jeder beliebige Wert verwendet werden, um die gesamte Nachricht zu kodieren. Zum Beispiel könnte der Wert 0.17 oder der Mittelwert $(0.166 + 0.18)/2 = 0.173$ sein.

Auf der Grundlage der verwendeten Häufigkeitstabelle wird die Nachricht **abc** also mit dem Wert 0.173 kodiert. Wenn sich die Häufigkeitstabelle ändert, dann ändert sich auch der Wert, der die Nachricht kodiert.

Einschub binäre Bruchzahlen

Wie drückt man 0.173 binär aus?

Arithmetische Codierung - Dekodierung

Die Eingaben für die Dekodierung sind:

- Der Einzelwert, der die Nachricht kodiert.
- Die Frequenztabelle. Sie muss mit der für die Kodierung verwendeten Tabelle identisch sein.
- Die Anzahl der Symbole in der ursprünglichen Nachricht.
    - Alternativ ein Zeichen für EOM (end of message).

Beispiel

In unserem Beispiel ist der Wert, der die Nachricht kodiert: 0.173

Die Häufigkeitstabelle lautet $a = 2, b = 7, c = 1$.

Die Anzahl der Symbole in der Nachricht ist 3.

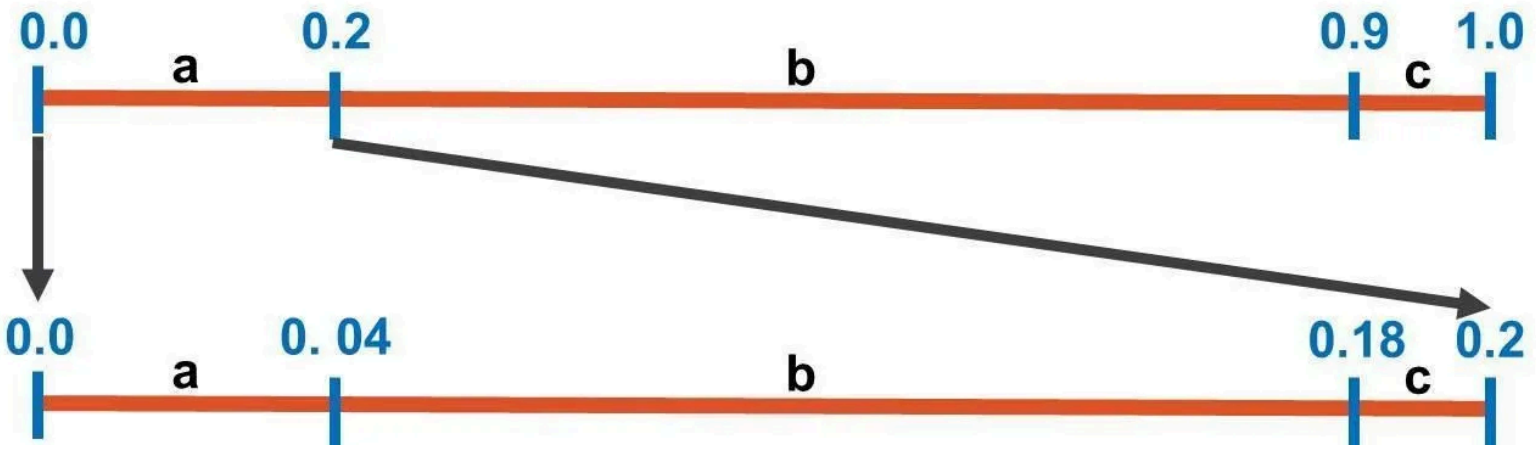
Die Dekodierung funktioniert ähnlich wie die Kodierung, indem dieselben Intervalle konstruiert werden.

Zunächst wird eine Linie erstellt, die bei 0.0 beginnt und bei 1.0 endet. Die Intervalle der 3 Symbole werden mit der gleichen Gleichung berechnet, die für die Kodierung der Nachricht verwendet wurde.

$$S_u = S_l + (P(C)) * R$$

Der Wert 0.173 fällt in das erste Intervall $[0.0, 0.2)$.

Das erste Symbol in der verschlüsselten Nachricht ist also **a**.

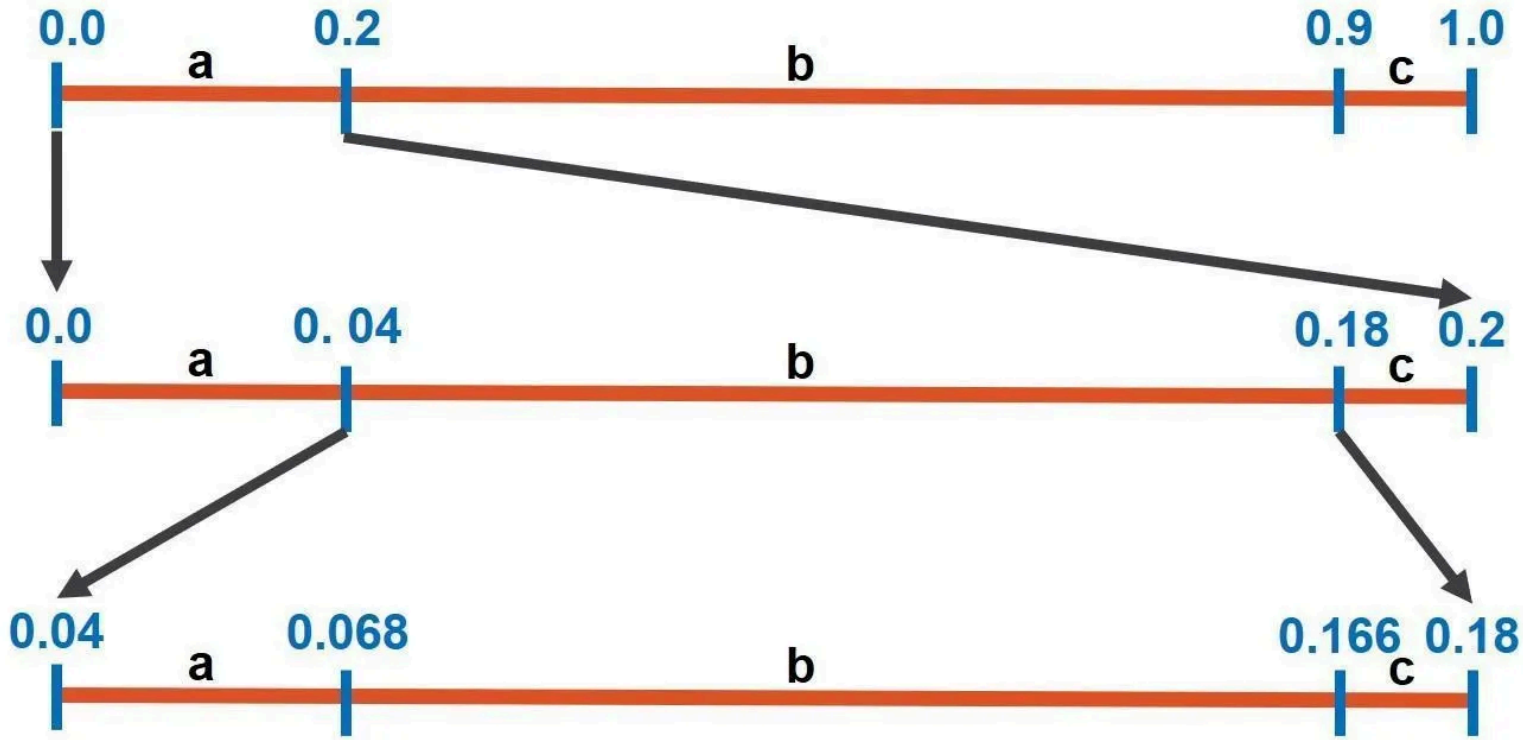


Und das nächste Intervall wird wieder wie bei der Kodierung unterteilt.

Der Wert 0.173 fällt jetzt in das zweite Intervall $[0.04, 0.18)$. Das zweite Symbol in der verschlüsselten Nachricht ist also **b**.

Im nächsten Schritt wird der Bereich auf das Intervall $[0.04, 0.18)$ eingeschränkt.

Der Wert 0.173 fällt jetzt in das dritte Intervall $[0.166, 0.18)$. Aus diesem Grund ist das nächste Symbol in der Nachricht **c**.



Da die Anzahl der Symbole in der ursprünglichen Nachricht 3 beträgt, ist der Dekodierungsprozess abgeschlossen. Die dekodierte Nachricht lautet **abc**.

Implementierung in Python 01

Wahrscheinlichkeiten berechnen und kumulieren

```
'''prepare arithmetic coding'''

# the example message "AABBBBBBBC" leads to the frequency
# dictionary {'A': 0.2, 'B': 0.7, 'C': 0.1} as used in the lecture
alphabeth_with_freq = "AABBBBBBBC"

# get the frequencies of the characters
frequencies = get_character_counts(alphabeth_with_freq)

# calculate the total number of characters
total = sum(frequencies.values())

# calculate probabilities
for char, freq in frequencies.items():
    frequencies[char] = freq / total

# calculate the cumulative frequencies
frequencies = dict(sorted(frequencies.items(), key=lambda item: item[0], reverse=False))
cumulative_frequencies = {}
cumulative_frequency = 0
for char, freq in frequencies.items():
    cumulative_frequencies[char] = cumulative_frequency
    cumulative_frequency += freq
```

Implementierung in Python 02

Kodierung

```
def encode(message, frequencies, \
           cumulative_frequencies, total):
    # initialize the interval
    low = 0
    high = 1
    # encode the message
    for char in message:
        # calculate the range
        range = high - low
        # calculate the new high and low values
        high = low + range * (cumulative_frequencies[char] \
         + frequencies[char]) / total
        low = low + range * cumulative_frequencies[char] / total
    return low
```

Implementierung in Python 03

Dekodierung mit bekannter Codelänge

```
def decode_with_length(encoded_message, frequencies, cumulative_frequencies, length):
    message = ""
    interval = encoded_message
    while len(message) < length:
        for char, freq in frequencies.items():
            if cumulative_frequencies[char] <= interval < cumulative_frequencies[char] + freq:
                message += char
                low = cumulative_frequencies[char]
                high = (cumulative_frequencies[char] + freq)
                interval = (interval - low) / (high - low)
                break
    return message
```

Problem der Implementierung

Welches Problem kann bei dieser Kodierung auftreten?

Alternative Implementierung in Python

ArithmeticEncodingPython von Ahmed Gad beinhaltet noch den korrekten Umgang mit der Präzision.