Bioinspirierte Codierung und Datenkompression WiSe 2012/13

Übungsblatt 1

Abgabe: 06.11.2012

Aufgabe 1 Entropie und Redundanz (25%)

Eine Quelle gibt Zeichen aus dem Alphabet $A = \{a_1, a_2, a_3, a_4, a_5\}$ mit den Wahrscheinlichkeiten $P(a_1) = 0.15$, $P(a_2) = 0.04$, $P(a_3) = 0.26$, $P(a_4) = 0.05$ und $P(a_5) = 0.5$ aus.

a)

Berechnet die Entropie der Quelle auf dem Papier (Für die Logarithmen etc. könnt Ihr natürlich einen Taschenrechner oder beliebige andere Hilfsmittel inklusive Tabellen und Computern verwenden, bei dieser Aufgabe kommt es nur darauf an, die Formel einmal selbst anzuwenden).

b)

Bestimmt einen Huffman Code für diese Quelle. Verwendet an dieser Stelle bitte hierfür kein Computerprogramm, sondern bestimmt den Code selbst, indem Ihr diesen wie in der Vorlesung gezeigt von Hand erzeugt.

c)

Bestimmt die durchschnittliche Codewortlänge für den Code in b) und berechnet die Redundanz.

Aufgabe 2 Unfaire Münzen und Informationsgehalt (25%)

Macht Euch das in der Vorlesung vorgestellte Konzept der Entropie an einem einfachen Beispiel klar: Betrachtet hierzu ein einfaches Münzwurf-Experiment, welches bekanntermaßen nur zwei unterschiedliche Resultate haben kann: Kopf oder Zahl. Berechnet zunächst die Entropie für den Wurf einer "fairen" Münze. Simuliert danach eine gezinkte Münze, indem Ihr die Wahrscheinlichkeit p für eines der Elementarereignisse in Schritten von 0.05 von 0.05 bis 0.95 variiert (Warum nicht zwischen 0 und 1?) und dem anderen Ereignis jeweils eine Wahrscheinlichkeit von 1-p zuweist. Bestimmt die Entropie eines Münzwurfexperiments für diese unterschiedlichen Wahrscheinlichkeiten und plottet die Resultate. (Tragt auf der x-Achse die Wahrscheinlichkeit für eines der Elementarereignisse und auf der y-Achse die dazugehörige Entropie auf.)

Was fällt auf? Bei welcher Wahrscheinlichkeit ist die Entropie des Experiments am höchsten? Wie ist dies zu erklären? Bitte gebt Eure Lösung inklusive dem Plot der Resultate und dem dazugehörigen Code ab.

Aufgabe 3 Huffman Codes (50%)

Implementiert ein Programm, welches für ein gegebenes Alphabet und den dazugehörigen Wahrscheinlichkeiten einen Huffman-Code mit minimaler Varianz der Codewortlängen generiert. Ach-

tet bei Eurer Implementierung darauf, dass diese möglichst generisch ist, da diese im Laufe des Semesters für unterschiedliche Symboltypen wiederverwendet werden soll.

Der Konstruktor Eurer Klasse bekommt ein Array übergeben, wobei jedes Array-Element ein Symbol darstellt. Aus dem Array sollen die Auftrittswahrscheinlichkeiten der Symbole berechnet und hiermit ein Huffman Code bestimmt werden.

Der Konstruktor soll weiter die Möglichkeit besitzen, einen übergebenen Code zu benutzen statt diesen zu generieren[1].

Eure Klasse soll außerdem die Funktionen `encode(array)` und `decode(string)` zum Kodieren bzw. Dekodieren anbieten. Bei der Kodierung soll das eingegebene Array (die zu codierenden Daten) dem Code entsprechend in eine neue Bitfolge umgewandelt werden, die Ihr hier einfach in einem String der Form `"011010"` ablegen könnt. Die Methode `decode` soll einen solchen String dekodieren, ihn also wieder dem Code entsprechend in ein Array umwandeln.

Testet Eure Implementierung mit dem in der ersten Übung behandelten Text[2]. Berechnet die Wahrscheinlichkeiten für das Auftreten einzelner Zeichen sowie die Wahrscheinlichkeiten für das Auftreten von möglichen 2er-Blöcken ebendieser Zeichen, wie dies in der Übung gezeigt wurde. Generiert mit dem von Euch implementierten Programm jeweils einen Huffman Code und vergleicht die durchschnittlichen Wortlängen, Entropien und Redundanzen der beiden resultierenden Codes und beschreibt und erklärt Eure Beobachtungen.

Optional könnt Ihr (zusätzlich) mit Hilfe eines regulären Ausdrucks alle Zeichen außer den Buchstaben von a-h durch Leerzeichen ersetzen und die Effizienz des resultierenden Codes mit der Effizienz des Codes vergleichen, den Ihr in der Übung von Hand erstellt habt.

[1] Unter Python ist das am einfachsten, wenn man in der `__init__` Methode Variablen mit *default*-Werten angibt und anschließend im Konstruktor auswertet, ob diese Variablen übergeben worden sind oder nicht.

[2] Zu finden unter `http://tools.ietf.org/rfc/rfc791.txt`