Sample solutions to Homework 3,
Information-Theoretic Modeling (Fall 2014)

Jussi Määttä

September 25, 2014

Question 1

(a)
Let

\[
\begin{align*}
\text{SET1} &= \text{the set of prefix(-free) codes}, \\
\text{SET2} &= \text{the set of decodable codes}, \\
\text{SET3} &= \text{the set of codes that satisfy the Kraft inequality,} \\
\text{SET4} &= \text{the set of all possible symbol codes.}
\end{align*}
\]

Then \(\text{SET1} \subseteq \text{SET2} \subseteq \text{SET3} \subseteq \text{SET4}.\)

(b)

- A code with codewords \(\{0, 01\}\) is not a prefix(-free) code, but it is decodable.
- A code with codewords \(\{0, 00\}\) is not decodable, but is satisfies the Kraft inequality: \(2^{-1} + 2^{-2} = 0.75.\)
- A code with codewords \(\{0, 1, 01\}\) is a symbol code, but it does not satisfy the Kraft inequality: \(2^{-1} + 2^{-1} + 2^{-2} = 1.25.\)
Question 2

1. Sort the symbols:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>E</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>$p_i$</td>
<td>0.9</td>
<td>0.04</td>
<td>0.02</td>
<td>0.015</td>
<td>0.015</td>
<td>0.01</td>
</tr>
</tbody>
</table>

2. Split into \{(x_1), (x_2, \ldots, x_6)\}:

- A 0
- C 1
- B 1
- E 1
- F 1
- D 1

The code for the symbol A is now ready.

3. Split (x_2, \ldots, x_6) into \{(x_2), (x_3, \ldots, x_6)\}. (Note that the split \{(x_2, x_3), (x_4, x_5, x_6)\} would be equally good.)

- A 0
- C 10
- B 11
- E 11
- F 11
- D 11

The codes for the symbols A and C are now ready.

4. Split (x_3, x_4, x_5, x_6) into \{(x_3, x_4), (x_5, x_6)\}.

- A 0
- C 10
- B 110
- E 110
- F 111
- D 111
5. The pairs \((x_3, x_4)\) and \((x_5, x_6)\) are can be split in only one way. The end result is the following:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1100</td>
</tr>
<tr>
<td>E</td>
<td>1101</td>
</tr>
<tr>
<td>F</td>
<td>1110</td>
</tr>
<tr>
<td>D</td>
<td>1111</td>
</tr>
</tbody>
</table>

(Note: had we chosen the split \(\{(x_2, x_3), (x_4, x_5, x_6)\}\) in step 3, the resulting codewords would be \(A = 0, C = 100, B = 101, E = 110, F = 1110, D = 1111\).)

The expected code-length for this particular code Shannon–Fano code is

\[
\sum_{i=1}^{6} \ell_i p_i = 1 \cdot 0.9 + 2 \cdot 0.04 + 4 \cdot (0.02 + 0.015 + 0.015 + 0.01)
\]

\[
= 1.22.
\]

The entropy of the source is

\[
H(X) = -\sum_{i=1}^{6} p_i \log_2 p_i \approx 0.6836
\]

and the expected code-length of the Shannon code for this source is

\[
E[\ell_{\text{Shannon}}(X)] = \sum_{i=1}^{6} p_i \left\lceil \log_2 \frac{1}{p_i} \right\rceil = 1.5.
\]

This is consistent with the known inequality

\[
E[\ell_{\text{Shannon}}(X)] \leq H(X) + 1.
\]
Question 3

The attached Python 3 program *shannon_fano.py* reads data from standard input and computes the desired quantities.

If we give it as input its own source code, we get the following:

\[
\begin{align*}
\text{entropy} &\approx 4.58, \\
\text{code-length} &\approx 4.60, \\
E[\text{code-length of the Shannon code}] &\approx 5.11.
\end{align*}
\]

The code-length is almost the same as the entropy, so this is a very good result. The Shannon code would not, in expectation, work as well.

Question 4

(a)

The binary tree given by the Huffman code is shown in Figure 1. We have always assigned the digit 0 to the left branch and the digit 1 to the right branch. One can read the codewords from the tree; for instance, \( B = 1100 \).

(b)

Consider a source \( X \) with the two-symbol alphabet \( \{a, b\} \), with \( \Pr[X = a] = 2^{-k} \) for some positive integer \( k \). Then

\[
\left\lceil \log_2 \frac{1}{2^{-k}} \right\rceil = k
\]

but the Huffman codewords for the symbols have length 1.

(c)

Consider the case where there are five symbols \( \{a, b, c, d, e\} \). If \( e \) has 2 occurrences, then after combining \( (a, b) \) with \( c \), the Huffman code will combine \( d \) with \( e \). But if \( e \) has 3 occurrences, then the algorithm faces a tie between combining \( (a, b, c) \) with \( d \), and \( d \) with \( e \); if we choose the former, we again get a maximally unbalanced Huffman tree.

What if there are six symbols? Then \( f \) must have at least 5 occurrences. For seven symbols, the number is 8. For eight symbols, it is 13.
Figure 1: The binary tree given by the Huffman code for the source in Exercise 2.

Let us denote the counts by \( c_1 = c_2 = c_3 = 1 \), \( c_4 = 2 \), \( c_5 = 3 \), \( c_6 = 5 \) and so on. We may assume that \( c_n \leq c_{n+1} \) for all \( n \), because the Huffman code sorts the symbols by frequency.

The key here is that the \( n \)'th symbol must have an occurrence count that is at least the sum of the counts of symbols \( 1, 2, \ldots, n-2 \). Why? Because that sum, \( S_{n-2} = \sum_{i=1}^{n-2} c_i \), is compared to the values \( c_{n-1} \) and \( c_n \), and to get a maximally unbalanced tree we must have \( c_n \geq S_{n-2} \) (otherwise, if \( c_n < S_{n-2} \), then the \( n \)'th and \( (n-1) \)'th nodes are combined with each other).

As we want to find the minimal values of \( c_n \), the solution to our question is the following:

\[
c_1 = c_2 = c_3 = 1,
\]

\[
c_n = \sum_{i=1}^{n-2} c_i \quad \text{for } n \geq 4.
\]
We now prove by induction that in fact $c_n = c_{n-1} + c_{n-2}$ for $n \geq 4$, that is, we have essentially the Fibonacci sequence! (Except for $c_1$.) First, note that this is satisfied for $n = 4$. Now,

$$c_{n+1} = \sum_{i=1}^{n-1} c_i = \sum_{i=1}^{n-2} c_i + c_{n-1} = c_n + c_{n-1}$$

so the claim is proven.

Suppose we have $m$ distinct source symbols with the above counts $c_1, \ldots, c_m$. The symbol $a$ occurs once and there are a total of $\sum_{i=1}^{m} c_i = c_{m+2}$ occurrences, so the probability of $a$ is $1/c_{m+2}$.

When there are $m$ symbols with these counts, the depth of the Huffman tree (equivalently, the codeword length for the symbol $a$) is $m - 1$. To see why, consider that when we start from the root of the tree, we must separately “decide” against every other symbol before we reach $a$. A rigorous argument can again be made by induction: the claim holds for $m = 4$, and adding a new node with weight $c_{m+1}$ must increase the depth of the tree by one.

The Shannon codeword length is

$$\left\lceil \log_2 \left( \frac{1}{p(a)} \right) \right\rceil = \left\lceil \log_2 c_{m+2} \right\rceil.$$

Since one can show that the Fibonacci numbers have the closed form\(^1\)

$$c_n = \frac{\varphi^{n-1} - (-\varphi)^{-(n-1)}}{\sqrt{5}}, \quad \varphi = \frac{1 + \sqrt{5}}{2} \approx 1.62,$$

we have that $c_n \approx \varphi^{n-1}/\sqrt{5}$ for large $n$ and hence

$$\left\lceil \log_2 c_{m+2} \right\rceil \leq 1 + \log_2 c_{m+2} \approx 1 + (m + 1) \log_2 \varphi - \log_2 \sqrt{5} \leq 0.7m + 0.6$$

for large $m$. This is asymptotically smaller than $m - 1$, so the Shannon codeword length of $a$ becomes smaller than the Huffman codeword length. (In fact, one may calculate numerically that the codelengths of $a$ are the same for $m = 2, 3, 4, 5$ and the Shannon codeword length is strictly smaller for $m \geq 6$.)

\(^1\)See e.g. http://mathworld.wolfram.com/BinetsFibonacciNumberFormula.html.
Question 5

First, if $\Pr[X = 0] = p = 0.5$, then it obviously suffices to always use exactly one fair coin flip, and the expected number of flips required is 1.

Suppose then that $p \neq 0.5$. Consider the following procedure:

Procedure 1:

1. Set $p_0 \leftarrow p$ and $p_1 \leftarrow 1 - p$.
2. Flip a fair coin. If it comes out heads, then
   
   (a) if $p_0 \geq p_1$, return 0,
   (b) if $p_0 < p_1$, return 1.
3. If $p_0 \geq p_1$, set $p_0 \leftarrow p_0 - 0.5$.
   Otherwise, set $p_1 \leftarrow p_1 - 0.5$.
4. Normalize $p_0$ and $p_1$ so that $p_0 + p_1 = 1$.
5. Go to step 2.

What does this procedure do? For example, consider the case $p_0 = 0.3$. Let’s see what happens when we first enter step 2. Take a look at Figure 2 to get an idea of what’s going on.

![Figure 2: The situation at the first iteration of Procedure 1](image)

We flip a fair coin. If it comes out heads, then we return 1. Otherwise, the situation is inconclusive: we have “consumed” 0.5 worth of probability mass
from the event $X = 1$ but it still has 0.2 probability mass left. Technically speaking, we are decomposing the probability of the event $X = 1$ as

$$
\Pr[X = 1] = \Pr[X = 1 \mid \text{heads}] \Pr[\text{heads}] + \Pr[X = 1 \mid \text{tails}] \Pr[\text{tails}]
$$

$$
= 1 \cdot \frac{1}{2} + \Pr[X = 1 \mid \text{tails}] \cdot \frac{1}{2}
$$

$$
= \frac{1}{2} + \frac{1}{2} \Pr[X = 1 \mid \text{tails}].
$$

So if the fair coin comes up heads (probability 0.5), we are done; if it comes up tails, we continue. The continuation goes on as shown in Figure 3.

This was the intuition behind the procedure. To analyze it mathematically, we first simplify it a little. We don’t really need to keep track of both $p_0$ and $p_1$, since $p_1 = 1 - p_0$. In step 2, we return 0 if $p_0 \geq 0.5$ and 1 otherwise. In step 4, the normalization constant is always

$$
\frac{1}{p_0 + p_1 - 0.5} = \frac{1}{p_0 + (1 - p_0) - 0.5} = \frac{1}{0.5} = 2.
$$

Having made these observations, we can rewrite the procedure as follows:

**Procedure 2:**

1. Flip a fair coin. If it comes out heads, then
   
   (a) if $p \geq 0.5$, return 0,
   
   (b) if $p < 0.5$, return 1.

2. If $p \geq 0.5$, set $p \leftarrow 2(p - 0.5) = 2p - 1$. Otherwise, set $p \leftarrow 2p$.

3. Go to step 1.

This looks much simpler! Let us make yet another observation. Recall that

$$
\sum_{i=1}^{\infty} 2^{-i} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = 1.
$$

Therefore, since $0 < p < 1$, we can write

$$
p = \sum_{i=1}^{\infty} b_i 2^{-i}, \quad b_i \in \{0, 1\},
$$
that is, the bits $b_i$ give a binary representation of $p$. It holds that $p \geq 0.5 \iff b_1 = 1$. And

$$2p = \sum_{i=1}^{\infty} b_i 2^{-i+1} = \begin{cases} \sum_{i=2}^{\infty} b_i 2^{-i+1} & \text{if } b_1 = 0, \\ 1 + \sum_{i=2}^{\infty} b_i 2^{-i+1} & \text{if } b_1 = 1, \end{cases}$$

from which we see that step 2 above simply means that we discard the first bit of $p$ (i.e., we do a one-step bit shift). The steps 1–3 go through the bit representation of $p$!

The outcome of our procedure is denoted by $X$. Let $T_i$ be the event that
the procedure terminates at the \( i \)'th coin flip. Then \( \Pr[X = 0 \mid T_i] = 1 \) if and only if, after \( i \) iterations, \( p \geq 0.5 \) or equivalently \( b_i = 1 \). By the above observations, we have

\[
\Pr[X = 0] = \sum_{i=1}^{\infty} \Pr[X = 0 \mid T_i] \Pr[T_i] = \sum_{i=1}^{\infty} b_i 2^{-i} = p
\]

so the procedure indeed produces the desired probability.

The expected number of fair coin flips that are required is

\[
E[\text{n:o of flips needed}] = \sum_{k=1}^{\infty} k \Pr[k \text{ flips needed}] = \sum_{k=1}^{\infty} k 2^{-k}.
\]

To see that this equals 2, consider the partial sums

\[
S_n = \sum_{k=1}^{n} \frac{k}{2^k} = \sum_{k=1}^{n} \frac{1 + (k - 1)}{2^k} = \sum_{k=1}^{n} 2^{-k} + \sum_{k=1}^{n} \frac{k - 1}{2^k} = \sum_{k=1}^{n} 2^{-k} + \frac{1}{2} \sum_{k=1}^{n} \frac{k}{2^k} = \sum_{k=1}^{n} 2^{-k} + \frac{1}{2} S_n.
\]

From the above, we may solve \( S_n = 2 \sum_{k=1}^{n} 2^{-k} \) which tends to 2 as \( n \to \infty \). (Another way to compute the expectation would be to notice that we’re dealing with what’s called the geometric distribution and use its well-known (to some) properties.)