

第一章 作业题

September 23, 2021

Problem 1 (5) 考虑表 1.1 中列出的信源符号和它们相应的概率。对于这个码，求信源的熵、每个符号的平均二元字节数、该码的效率。

符号	概率	自信息	码字
x_1	0.40	1.3219	1
x_2	0.35	1.5146	00
x_3	0.25	2.0000	01

Table 1.1: 信源符号和它们相应的概率、自信息及码字

Solution. 信源的熵:

$$\begin{aligned}
 H(X) &= -\sum_{i=1}^3 P(x_i) \log_2 P(x_i) \\
 &= -x_1 \log_2(x_1) - x_2 \log_2(x_2) - x_3 \log_2(x_3) \\
 &= 0.40 \times \log_2\left(\frac{1}{0.4}\right) + 0.35 \times \log_2\left(\frac{1}{0.35}\right) + 0.25 \times \log_2\left(\frac{1}{0.25}\right) \\
 &= 1.5589 \text{ (bit)}
 \end{aligned}$$

每个符号的平均二元字节数:

$$\begin{aligned}
 \bar{R} &= \sum_{i=1}^3 n_i P(x_i) \\
 &= 1 \times 0.40 + 2 \times 0.35 + 2 \times 0.25 \\
 &= 1.60 \text{ (bit)}
 \end{aligned}$$

该码的效率:

$$\begin{aligned}
 R &= \frac{H(U)}{\bar{R}} \\
 &= \frac{1.5589}{1.60} \\
 &= 0.9743
 \end{aligned}$$

Problem 2 (12) 为什么要用信道编码器？信道编码要满足什么定理？请给出该定理的内容。

Solution. 为了使传输有效。信道编码器将信息序列 U 变换成离散的有结构的编码序列 X ，这称为码字。即为了使传输有效，人为的增加一些冗余度，使其具有自动检错和纠错的能力。码字的结构主要用以对付传输或储存码字的有扰信道。

信道编码要满足信道编码定理 (仙农第二定理)。信道编码定理是阐明使传信率逼近信道容量大编码是存在的定律。如果系统的传输率小于信道容量，那么适当选择编码技术就能实现可靠通信，即可以将差错率减小到任意小的程度。每个信道都具有固定的信道容量 C ，对任何小于 C 对信息传输率 R ，存在一个码长为

n 码率为 R 对分组码。若用最大似然译码, 则其译码错误概率为 $P_e \leq \Lambda e^{-nE_b(R)}$ 。对于码率为 R 约束长度为 n_c 对卷积码, 其译码错误概率也有类似的关系。其中 A 和 B 都为大于 0 对数, $E_b(R)$ 和 $E_c(R)$ 为正实函数, 叫做误差函数。

Problem 3 (18) 对于如图 3.1 所示的 BSC 信道, 信源符号发生的概率为 $P(x_1) = 0.6, P(x_2) = 0.4$ 。求:

1. 信源 X 中事件 x_1 和 x_2 分别的自信息 (以比特为单位);
2. 接收符号 $y_i (i = 1, 2)$ 发生的概率;
3. 求条件概率 $P(x_i|y_i)$;
4. 收到消息 $y_i (i = 1, 2)$ 后, 获得的关于 $x_i (i = 1, 2)$ 的信息量;
5. 信源 X 和信源 Y 的信息熵;
6. 条件熵 $H(X|Y)$ 和 $H(Y|X)$ 。

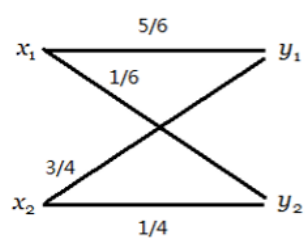


Figure 3.1: 二元 BSC 信道

Solution. 1. 信源 X 中事件 x_1 和 x_2 分别的自信息 (以比特为单位);

$$I(x_1) = \log \frac{1}{P(x_1)} = \log \frac{1}{0.6} = 0.737 \text{ bits}$$

$$I(x_2) = \log \frac{1}{P(x_2)} = \log \frac{1}{0.4} = 1.322 \text{ bits}$$

2. 接收符号 $y_i (i = 1, 2)$ 发生的概率;

$$\begin{aligned} P(y_1) &= P(x_1) \times P(y_1|x_1) + P(x_2) \times P(y_1|x_2) \\ &= 0.6 \times \frac{5}{6} + 0.4 \times \frac{3}{4} \\ &= 0.8 \end{aligned}$$

$$\begin{aligned} P(y_2) &= P(x_1) \times P(y_2|x_1) + P(x_2) \times P(y_2|x_2) \\ &= 0.6 \times \frac{1}{6} + 0.4 \times \frac{1}{4} \\ &= 0.2 \end{aligned}$$

3. 求条件概率 $P(x_i|y_i)$;

$$\begin{aligned}
 P(x_1|y_1) &= \frac{P(x_1)P(y_1|x_1)}{\sum_{j=1}^2 P(x_j)P(y_1|x_j)} \\
 &= \frac{0.6 \times \frac{5}{6}}{0.6 \times \frac{5}{6} + 0.4 \times \frac{3}{4}} \\
 &= 0.625
 \end{aligned}$$

$$\begin{aligned}
 P(x_2|y_1) &= \frac{P(x_2)P(y_1|x_2)}{\sum_{j=1}^2 P(x_j)P(y_1|x_j)} \\
 &= \frac{0.4 \times \frac{3}{4}}{0.6 \times \frac{5}{6} + 0.4 \times \frac{3}{4}} \\
 &= 0.375
 \end{aligned}$$

$$\begin{aligned}
 P(x_1|y_2) &= \frac{P(x_1)P(y_2|x_1)}{\sum_{j=1}^2 P(x_j)P(y_2|x_j)} \\
 &= \frac{0.6 \times \frac{1}{6}}{0.6 \times \frac{1}{6} + 0.4 \times \frac{1}{4}} \\
 &= 0.5
 \end{aligned}$$

$$\begin{aligned}
 P(x_2|y_2) &= \frac{P(x_2)P(y_2|x_2)}{\sum_{j=1}^2 P(x_j)P(y_2|x_j)} \\
 &= \frac{0.4 \times \frac{1}{4}}{0.6 \times \frac{1}{6} + 0.4 \times \frac{1}{4}} \\
 &= 0.5
 \end{aligned}$$

4. 收到消息 $y_i (i = 1, 2)$ 后, 获得的关于 $x_i (i = 1, 2)$ 的信息量;

$$\begin{aligned}
 I(x_1|y_1) &= -\log_2 P(x_1|y_1) \\
 &= -\log_2(0.625) \\
 &= 0.67807
 \end{aligned}$$

$$\begin{aligned}
 I(x_2|y_1) &= -\log_2 P(x_2|y_1) \\
 &= -\log_2(0.375) \\
 &= 1.41503
 \end{aligned}$$

$$\begin{aligned}
 I(x_1|y_2) &= -\log_2 P(x_1|y_2) \\
 &= -\log_2(0.5) \\
 &= 1
 \end{aligned}$$

$$\begin{aligned}
 I(x_2|y_2) &= -\log_2 P(x_2|y_2) \\
 &= -\log_2(0.5) \\
 &= 1
 \end{aligned}$$

5. 信源 X 和信源 Y 的信息熵;

$$\begin{aligned}
 H(X) &= -\sum_{i=1}^2 P(x_i) \log_2 P(x_i) \\
 &= -0.6 \times \log_2(0.6) - 0.4 \times \log_2(0.4) \\
 &= 0.97095 \text{ (bit)}
 \end{aligned} \tag{3.1}$$

$$\begin{aligned}
 H(Y) &= -\sum_{i=1}^2 P(y_i) \log_2 P(y_i) \\
 &= -0.8 \times \log_2(0.8) - 0.2 \times \log_2(0.2) \\
 &= 0.72193 \text{ (bit)}
 \end{aligned} \tag{3.2}$$

6. 条件熵 $H(X|Y)$ 和 $H(Y|X)$ 。

$$\begin{aligned}
 H(X|Y) &= \sum_{i=1}^2 \sum_{j=1}^2 P(x_i y_j) \log_2 \frac{1}{P(x_i|y_j)} \\
 &= -\sum_{i=1}^2 \sum_{j=1}^2 P(x_i y_j) \log_2 P(x_i|y_j) \\
 &= -\frac{5}{6} \times \log_2\left(\frac{5}{6}\right) - \frac{1}{6} \times \log_2\left(\frac{1}{6}\right) - \frac{3}{4} \times \log_2\left(\frac{3}{4}\right) - \frac{1}{4} \times \log_2\left(\frac{1}{4}\right) \\
 &= 1.46130
 \end{aligned}$$

$$\begin{aligned}
 H(Y|X) &= \sum_{i=1}^2 \sum_{j=1}^2 P(y_i x_j) \log_2 \frac{1}{P(y_i|x_j)} \\
 &= -\sum_{i=1}^2 \sum_{j=1}^2 P(y_i x_j) \log_2 P(y_i|x_j) \\
 &= -0.625 \times \log_2(0.625) - 0.375 \times \log_2(0.375) - 0.5 \times \log_2(0.5) - 0.5 \times \log_2(0.5) \\
 &= 1.95443
 \end{aligned}$$

Problem 4 (19) 上机题。

1. 如何编程序实现霍夫曼编码?

Solution. huffmanTree.h

```

1 #include <iostream>
2 #include <queue>
3 #include <map>
4 #include <string>

```

```
5
6 using namespace std;
7
8 namespace HuffmanTree {
9
10     struct Node{
11         char c;
12         int frequency;
13         Node *left;
14         Node *right;
15
16         Node(char _c, int _frequency, Node *_left = nullptr, Node *_right = nullptr)
17             : c(_c), frequency(_frequency), left(_left), right(_right) {}
18
19         bool operator<(const Node &node) const {
20             return frequency > node.frequency;
21         }
22     };
23
24     class huffmanTree
25     {
26     private:
27         std::priority_queue<Node> pq;
28
29         void _huffmanCode(Node *node,
30                             std::string &prefix,
31                             std::map<char,
32                             std::string>& codeMap) {
33             std::string tmp = prefix;
34             if (node->left != nullptr) {
35                 prefix += '0';
36                 if (!_isLeaf(node->left)) {
37                     codeMap[node->left->c] = prefix;
38                 } else {
39                     _huffmanCode(node->left, prefix, codeMap);
40                 }
41             }
42             if (node->right != nullptr) {
43                 prefix = tmp;
44                 prefix += '1';
45                 if (!_isLeaf(node->right)) {
46                     codeMap[node->right->c] = prefix;
47                 } else {
48                     _huffmanCode(node->right, prefix, codeMap);
49                 }
50             }
51         }
52
53         static bool _isLeaf(Node* node) {
54             return node->left == nullptr && node->right == nullptr;
55         }
56
57     public:
58         huffmanTree(const std::map<char, int> alphabet) {
```

```
59         for(auto x : alphabet) {
60             Node node(x.first, x.second);
61             pq.push(node);
62         }
63         genHuffmanTree();
64     }
65     ~huffmanTree() {
66
67     }
68     void genHuffmanTree() {
69         while(pq.size() != 1) {
70             Node *left = new Node(pq.top());
71             pq.pop();
72             Node *right = new Node(pq.top());
73             pq.pop();
74             Node node('.', left->frequency + right->frequency, left, right);
75             pq.push(node);
76         }
77     }
78     void huffmanCode(std::map<char, std::string> &codeMap) {
79         Node node = pq.top();
80         std::string prefix;
81         _huffmanCode(&node, prefix, codeMap);
82     }
83 };
84 }
```

main.cpp

```
1  #include "huffmanTree.h"
2
3  using namespace HuffmanTree;
4
5  int main() {
6      std::map<char, int> _alphabet =
7          {{'a', 5}, {'b', 4}, {'c', 3}, {'d', 2}, {'e', 1}};
8      std::map<char, std::string> _codeMap;
9
10     huffmanTree tree(_alphabet);
11     tree.huffmanCode(_codeMap);
12
13     for(auto x : _codeMap) {
14         printf("%c: %s\n", x.first, x.second.c_str());
15     }
16
17     return 0;
18 }
```

output:

- a: 11
- b: 10
- c: 00

- d: 011
 - e: 010
-