信息论 Lecture Note 6 EX

（这一部分不考，就不写英文了）

随机变量生成

问题

给定一个离散随机变量 X，要投硬币（均匀的）多少次才能生成它的概率分布？

建模

我们建立一个映射 $f: Z \mapsto \mathcal{X}$，将投硬币得到的二进制序列映射成为字母表中的元素。那么对于 $p_i = p(X = x_i)$，只需要满足 $\sum_{f(z)=x_i} p(Z = z) = p_i$ 就行。设随机变量 T 表示序列长度。

我们还可以将二进制序列映射到一棵二叉树上。这样给每一个叶子分配一个 $\mathcal{X}$ 中的元素，就代表从根到该叶子的二进制序列映射到该元素上。

方便起见，我们保证不存在未被分配的叶子。则这个树有以下性质：

1. 树中节点要么有两个儿子，要么是叶子。也称树是完全的（Complete）。
2. 深度为 k 的叶子被取到的概率为 2^{-k}。
3. 树的期望深度等于 ET。

树可能是无限的（大概能猜到）。

基础情况

如果对于一棵完全二叉树，把它所有叶子的概率取出来得到一个分布 Y，那么显然有 $H(Y) = ET$。

定理 1：$\forall X, H(X) \le ET$。

一句话证明：显然 Y 是和叶子一一对应的。我们可以构建一个函数 $g: \mathcal{Y} \mapsto \mathcal{X}$，从而使得这些 Y 对应的叶子的概率加和等于 X 的概率。从而 $H(X) = H(g(Y)) \le H(Y) = ET$。

特别的，如果 X 是二进制的，那么可以取等号。原因在于此时构建的哈夫曼码和香农码具有相同的编码长度，且期望长度都达到了 $H(X)$。

扩展情况

如果 X 不是二进制的，可以对 X 的概率求二进制展开式，然后将展开的二进制项都映射到该概率值对应的字符即可。

界

定理 2：对于任意的 X，$H(X) \le ET < H(X) + 2$。

该定理证明比较复杂，在此略去。

通用信源编码

问题

很多时候都不知道具体的分布，那么，采用什么样的编码方式才能尽可能的优秀呢？

建模

假定随机变量 X 服从分布簇 $\{P_\theta\}$ 中的某个分布，其中参数 $\theta \in I$ 未知，I 是指标集。

假设我们使用的编码长度为 $l(x)$，概率为 $q(x) = 2^{-l(x)}$。定义**编码的冗余度** $R(p_\theta, q)$ 为

$$
\begin{aligned}
R\left(p_\theta, q\right) &= E_{p_\theta}[l(X)] - E_{p_\theta}\left[\log \frac{1}{p_\theta(X)}\right] \\
&= \sum_x p_\theta(x)\left(l(x) - \log \frac{1}{p(x)}\right) \\
&= \sum_x p_\theta(x)\left(\log \frac{1}{q(x)} - \log \frac{1}{p(x)}\right) \\
&= \sum_x p_\theta(x) \log \frac{p_\theta(x)}{q(x)} \\
&= D\left(p_\theta \| q\right)
\end{aligned}
$$

注意，这里假定严格按照 p_θ 做最优编码时，长度都是整数。

无论真实分布如何，都希望我们用的编码表现很好。于是定义**最小最大冗余度**（minimax redundancy）为

$$
R^* = \min_q \max_{p_\theta} R(p_\theta, q) = \min_q \max_{p_\theta} D(p_\theta \| q)
$$

定理 3：将分布簇写做转移矩阵形式（即每一个分布作为行向量，叠成一个矩阵），从而将 θ、这个矩阵和 $\mathcal{X}$ 构成一个信道（直观理解成确定 θ 得到 X 的分布）。则这个信道的容量等于 R^*。

该定理证明比较复杂，在此略去。

下面我们给出一些编码方案。

Shannon-Fano-Elias 编码

也称为算术编码（Arithmetic Coding）。它是一个对序列进行编码的方案，在之前的分析中已经出现过。

它将一个序列化成一个被包含于 $[0, 1)$ 的区间，使用区间的**中点**作为码字，区间长度作为概率生成码长。这里以一个例子阐释编码方式。

举例

假设字母表为 $\{A, B, C\}$，概率分布为 $\{0.4, 0.4, 0.2\}$。现在编码 $ACAA$。从 $[0, 1)$ 作为起始区间。

到 A：$[0, 1) \to [0, 0.4)$。

到 C：$[0, 0.4) \to [0.32, 0.4)$。

到 A：$[0.32, 0.4) \to [0.32, 0.352)$。

到 A：$[0.32, 0.352) \to [0.32, 0.3328)$。

中点是 0.3264，区间长为 0.0128，编码长度就使用 $-\lceil \log 0.0128 \rceil + 2$，即 9 bit。

归纳

从上面的编码方案可以看出：如果当前区间为 $[x, y)$，当前字符所在的分布为 $[a, b)$，那么更新后的区间就是 $[x + (y - x)a, x + (y - x)b)$。

结论

这样编码的长度和二进制编码的界是相同的，由此前的分析可知。

Limper-Ziv 压缩算法

也叫做 LZW 算法（好像是因为还有一个人名）。

这个算法有两个版本，一个用到的是滑动窗口（Sliding Window），一个用到的是字典树（Trie）。

滑动窗口版本

给定串 $S[0,l)$，希望将其压缩。目标在于将其分割成尽量少的段，且**串后面的段能用前面的段表示。**

设初始变量 $i=0$（表示开始切分的位置），滑窗大小为 W。执行下面的算法：

1. 找到 j,k 使得 $i-W\le j<i$ 的情况下，最大化满足 $S[j,j+k)=S[i,i+k)$ 的 k。
2. 如果这样的 k 不存在，就将当前字符切分出来，令 i 自增 1；否则将 $S[i,i+k)$ 切分出来，令 $i:=i+k$，回到步骤 1，直到 $i=l$ 停止。

每一次切分的结果都可以表示为一个元组。如果没有找到匹配（即 k 不存在），结果就是 $(0,S[i])$；否则结果为 $(1,j,k)$，表示匹配的起始位置是 j，匹配上的长度为 k。最后将所有元组连起来得到一个元组序列，利用该序列可以做到将压缩的信息恢复。恢复的过程较为简单，不再赘述。

可以看出元组的第一个元素是标志位，表明是否找到了匹配。需要注意的是，储存匹配的起始位置时有两种方式：保存串上的绝对位置和窗口中的相对位置。一般设定匹配起始位置 j 相对于切分点 i 的相对位置为 $i-j$，也就是从窗口的末端向前开始测量。

作为例子，考虑串 `ABBABBABBBAABABA`。切分出的串为 `A`，`B`，`B`，`ABBABB`，`BA`，`A`，`BA`，`BA`，得到的元组序列是 $(0,A),(0,B),(1,1,1),(1,3,6),(1,4,2),(1,1,1),(1,3,2),(1,2,2)$（这里采用的是相对位置）。

字典树版本

给定串 $S[0,l)$，希望将其压缩。目标在于将其分割成尽量少的段，且**尽可能利用串后面的段和前面的段相交的部分。**

算法的切分原则是将串顺序分解为**直到目前还未出现过的最短的字符串。**

设初始变量 $i=0$，分割出来的串的集合为 T，初始 $T=\emptyset$。执行下面的算法：

1. 最小化 k 使得 $0<k\le l-i$ 且 $S[i,i+k)$ 不在 T 中。
2. 如果不存在合法的 k 就停止算法；否则把 $S[i,i+k)$ 加入 T，令 $i:=i+k$，回到步骤 1，直到 $i=l$ 停止。

容易发现，任何时候，对于我们分割出的串，它的所有前缀（除了它自己）都已经在 T 中。因此集合 T 可用前缀树（Trie）维护。

同样地，每一次切分的结果都可以表示为一个元组 (st,c)，其中 c 表示切分出的串的最后一个字符，st 表示切分出的串去掉末尾的 c 得到的前缀在原串中的起始位置，如果串只包含 c 这一个字符 st 就是 0。恢复原串的方法和切分过程类似。

作为例子，考虑序列 `ABBABBABBBAABABA...`（这里不考虑不存在合法切分的边界情况）。切分出的串为 `A`，`B`，`BA`，`BB`，`AB`，`BBA`，`ABA`，`BAA`，`...`，得到的元组序列为 $(0,A),(0,B),(2,A),(2,B),(1,B),(4,A),(5,A),(3,A)$。