Tema 1
Teoría de la información

1 Medida de la información

Podemos modelar la salida de cualquier fuente de información (voz, vídeo, datos, etc.) como un proceso aleatorio. Para el caso de procesos aleatorios discretos, estacionarios y sin memoria, se puede definir la cantidad de información como:

$$I = -log_2(p_k)$$

donde p_k representa la probabilidad de aparición de cada uno de los símbolos de la fuente.

De forma similar se puede definir la **entropía** de la fuente como:

$$H = -\sum_{k=0}^{N-1} p_k \cdot log_2(p_k) \tag{1}$$

Habitualmente la entropía se mide en bits, y nos da una idea de cuántos bits por símbolo como mínimo se necesitan para codificar una determinada fuente de forma exacta.

2 Codificación de fuente

La codificación de fuente, o codificación sin ruido, se refiere al conjunto de técnicas que permiten reducir el número de bits necesarios para representar la salida de una fuente sin perder la posibilidad de reconstrucción perfecta.

El teorema de codificación de fuente de Shannon nos dice que, para reconstruir perfectamente una fuente, **es posible utilizar un código con un número de bits por símbolo nunca inferior a la entropía de la fuente**.

En la práctica existen distintos algoritmos de codificación de fuente, pero nosotros vamos a centrarnos sólo en los códigos Huffman, creados por David Huffmann en 1951 cuando preparaba un trabajo para una asignatura de doctorado en el MIT, donde estudiaba.

2.1 Codificación Huffman

La idea que hay detrás de los **códigos Huffman** es muy sencilla: se asignan palabras con más bits a aquellos símbolos de la fuente menos probables, y palabras más cortas a los símbolos más probables. Para hacerlo, empezamos uniendo los dos símbolos menos probables para generar un nuevo símbolo cuya probabilidad de aparición será la suma de las dos probabilidades anteriores. Este proceso se repite hasta que sólo quede un símbolo, y de esta forma se va generando un árbol.

Ahora, empezando desde la raíz del árbol, vamos asignando 0s y 1s a cualesquiera dos ramas que surjan del mismo nodo, y de esta manera se irá creando el código.

Vamos a verlo con un ejemplo muy sencillo. Imaginemos una fuente con un alfabeto de nueve símbolos con las siguientes probabilidades:

$$\mathbf{p} = (0.2, 0.15, 0.13, 0.12, 0.1, 0.09, 0.08, 0.07, 0.06)$$

En la primera iteración, agrupamos los dos símbolos menos probables, generando un nuevo símbolo con probabilidad $0.07 + 0.06 = 0.13$. Ordenamos el nuevo vector de probabilidades, obteniendo ahora:

$$\mathbf{p} = (0.08, 0.09, 0.1, 0.12, 0.13, 0.13, 0.15, 0.2)$$

Repetimos el proceso, agrupando los dos símbolos menos probables, obteniendo un nuevo símbolo con probabilidad $0.08 + 0.09 = 0.17$, y por tanto un nuevo vector de probabilidades que, tras ordenarlo, queda:

$$\mathbf{p} = (0.1, 0.12, 0.13, 0.13, 0.15, 0.17, 0.2)$$

Si seguimos iterando este algoritmo, en los siguientes pasos iremos obteniendo los siguientes vectores de probabilidades de error:

$$\mathbf{p} = (0.13, 0.13, 0.15, 0.17, 0.2, 0.22)$$

$$\mathbf{p} = (0.15, 0.17, 0.2, 0.22, 0.26)$$

$$\mathbf{p} = (0.2, 0.22, 0.26, 0.32)$$

$$\mathbf{p} = (0.26, 0.32, 0.42)$$

$$\mathbf{p} = (0.42, 0.58)$$

Ahora viene el momento de ir retrocediendo. Empezamos asignando un 0 al símbolo con probabilidad 0.42, y un 1 al otro. En el siguiente paso, el símbolo con probabilidad 0.58 se descompone en dos, con probabilidades 0.26 y 0.32. Así que al 1 de antes, le añadimos un 0 en el primer caso, y un 1 en el segundo. Tenemos por tanto que el símbolo con probabilidad 0.26 se codifica con un 10, el de probabilidad 0.32 con 11 y el de probabilidad 0.42 con 0.

Igual que antes, vamos repitiendo este proceso hasta llegar a los 9 símbolos originales, obteniéndose este resultado:

Símbolo	Probabilidad	Código
s_1	0.2	00
s_2	0.15	110
s_3	0.13	101
s_4	0.12	011
s_5	0.1	010
s_6	0.09	1111
s_7	0.08	1110
s_8	0.07	1001
s_9	0.06	1000

Para evaluar la calidad de un código fuente se pueden utilizar distintos parámetros, aunque nosotros nos vamos a fijar sólo en dos: la tasa de compresión y la eficiencia.

Definimos **longitud media** de un código como la longitud promedio de las palabras del mismo:

$$\bar{L} = \sum_{i=1}^{M} p_i n_i \tag{2}$$

donde n_i es el número de bits utilizados para codificar el símbolo i-ésimo.

Se puede comprobar de forma muy sencilla que la longitud media de palabra para este código es:

$$\bar{L} = 2 \cdot 0.2 + 3 \cdot (0.15 + 0.13 + 0.12 + 0.1) + 4 \cdot (0.09 + 0.08 + 0.07 + 0.06) = 3.1$$

A partir de la longitud media podemos definir la **tasa de compresión** como la relación de compresión lograda frente a un código de longitud fija:

$$\Gamma = \frac{\lceil log_2 M \rceil}{\bar{L}} \tag{3}$$

Por otro lado, la **eficiencia** de un código mide lo cerca que se encuentra su longitud media del límite teórico dado por la entropía:

$$\eta = \frac{H(x)}{\bar{L}} \tag{4}$$

Lógicamente, según el teorema de Shannon, será imposible obtener valores de la eficiencia superiores a 1.
Es posible demostrar que la longitud media de palabra de un código Huffman cumple la siguiente desigualdad:

$$H \leq \bar{L} \leq H + 1 \tag{5}$$

Y si agrupamos los símbolos en grupos de K, en lugar en enviarlos de forma individual, se obtiene que:

$$H \leq \bar{L} \leq H + \frac{1}{K} \tag{6}$$

Es decir, al incrementar el valor de K nos vamos a poder acercar cada vez más al valor de la entropía, aunque a costa, eso sí, de incrementar la complejidad del sistema de forma importante.

2.2 Codificación LZW

Uno de los inconvenientes de los códigos Huffman es que, para generarlos, necesitamos conocer las probabilidades de ocurrencia de cada uno de los símbolos, lo que no siempre es posible, aunque se pueden realizar estimaciones.

Unos de los códigos más utilizados es el LZW, que surgió como una versión del algoritmo LZ78 desarrollado por Abraham Lempel y Jacob Ziv, y mejorado por Terry Welch (el nombre del algoritmo viene de las iniciales de cada uno de sus creadores).

Este método de codificación fue el estándar del comando *compress* de Unix durante muchos años, hasta que se dejó de utilizar por algunos problemas de patentes. Asimismo, forma parte del formato de imagen GIF y también se puede utilizar para reducir el tamaño de los archivos TIFF.

En forma de pseudocódigo, podemos escribir el algoritmo de la siguiente forma:

```
CADENA = cadena vacía
WHILE queden caracteres por codificar DO
     CARACTER = coger el siguiente carácter
     IF CADENA+CARACTER está en el diccionario
          CADENA = CADENA+CARACTER
     ELSE
          código correspondiente a CADENA -> SALIDA
          Añadir CADENA+CARACTER al diccionario
          CADENA = CARACTER
     END
END
código para CADENA -> SALIDA
```

Imaginemos que tenemos un diccionario que tiene las siguientes entradas:

Código	Entrada
0	a
1	b
2	n

Y ahora supongamos que la cadena de entrada al codificador es *banana*. La codificación sería:

CADENA	CARACTER	¿En el Diccionario?	Al Diccionario	Salida
	b	Si		
b	a	No	3 - ba	1
a	n	No	4 - an	0
n	a	No	5 - na	2
a	n	Sí		
an	a	No	6 - ana	4
a				0

Es decir, la señal codificada sería: `1 0 2 4 0`.

La decodificación, por su parte, se hace de la siguiente forma:

```
CODIGO_1 = Leer primer código del mensaje
Traducción de CODIGO_1 -> SALIDA
WHILE queden caracteres por decodificar
     CODIGO_2 = Leer siguiente código
     CADENA = traducción de CODIGO_2
     CADENA -> SALIDA
     CARACTER = Primer carácter de CADENA
     Añadir (Traducción de CODIGO_1)+(CARACTER) al diccionario
     CODIGO_1 = CODIGO_2
END
```

CODIGO1	CODIGO2	CADENA	CARACTER	Salida	Dicc.
1				b	
1	0	a	a	a	3 - ba
0	2	n	n	n	4 - an
2	4	an	a	an	5 - na
4	0	a	a	a	6 - ana

Se puede ver que la salida es la misma palabra que había a la entrada y que se ha generado también el mismo diccionario que se generó en el proceso de codificación.