Introducción a la compresión de datos

La compresión se encuentra prácticamente en todas partes. Todas las imágenes que se reciben en la web están comprimidas, usualmente en formatos como JPEG o GIF; la mayoría de los módems utilizan compresión; la televisión en alta definición (HDTV) se transmite comprimida mediante MPEG-2; e incluso algunos sistemas de archivos comprimen automáticamente los archivos al almacenarlos, mientras que el resto de nosotros lo hacemos manualmente. Lo interesante de la compresión, como ocurre con otros temas abordados en este curso, es que los algoritmos que se emplean en el mundo real se apoyan en un conjunto amplio de herramientas algorítmicas, como ordenación, tablas hash, árboles trie y transformadas rápidas de Fourier (FFT). Además, los algoritmos con bases teóricas sólidas desempeñan un papel fundamental en aplicaciones prácticas.

En este contexto, utilizaremos el término genérico "mensaje" para referirnos a los objetos que queremos comprimir, que pueden ser archivos o mensajes propiamente dichos. La compresión consta de dos componentes: un algoritmo de codificación, que toma un mensaje y genera una representación comprimida (idealmente más corta en bits), y un algoritmo de decodificación, que reconstruye el mensaje original o una aproximación a partir de la representación comprimida. Ambos componentes deben estar íntimamente ligados, ya que deben compartir la comprensión del formato comprimido.

Distinguimos entre algoritmos sin pérdida, que permiten reconstruir exactamente el mensaje original, y algoritmos con pérdida, que solo reconstruyen una aproximación del mensaje. Los algoritmos sin pérdida se utilizan principalmente para texto, mientras que los algoritmos con pérdida se emplean con imágenes y sonidos, donde cierta pérdida de resolución puede ser imperceptible o aceptable. Es importante señalar que “con pérdida” no implica pérdida aleatoria de píxeles, sino la pérdida de alguna cantidad como un componente de frecuencia o incluso ruido. Por ejemplo, alguien podría pensar que la compresión con pérdida del texto es inaceptable, imaginando que implica caracteres intercambiados o desaparecidos. Pero si un sistema reformula frases a una forma más estándar o sustituye palabras por sinónimos para facilitar la compresión, técnicamente estaría aplicando compresión con pérdida, aunque el significado y la claridad del mensaje podrían conservarse, o incluso mejorar. De hecho, algunos dirían que escribir bien es una forma de compresión con pérdida del lenguaje.

¿Existe un algoritmo sin pérdida que pueda comprimir todos los mensajes? Se ha llegado a presentar una solicitud de patente afirmando que podía comprimir cualquier archivo si se aplicaba de forma recursiva, reduciéndolo casi a nada (Patente 5,533,051: "Métodos para la compresión de datos"). Sin embargo, con un poco de reflexión se puede comprobar que esto no es posible si los mensajes de entrada pueden contener cualquier secuencia de bits. Un simple argumento de conteo lo demuestra: por ejemplo, hay 210002^{1000}21000 mensajes diferentes de 1000 bits. Todos deben representarse de forma única para que el decodificador los distinga. No podemos representar esa cantidad con mensajes de 999 bits o menos, ya que con 999 bits solo podríamos distinguir 29992^{999}2999 mensajes. Si un mensaje se acorta mediante un algoritmo, entonces otro debe alargarse. Incluso se puede demostrar que, si se comprime uno de los mensajes posibles de una longitud fija, la longitud promedio de los mensajes comprimidos será superior a la original. Por ejemplo, si uno de los ocho posibles mensajes de 3 bits se comprime a 2 bits, al menos dos otros deben expandirse a 4 bits para que todos sigan siendo

distinguibles, lo que da una media de 3+1/83 + 1/83+1/8 bits. A pesar de ello, la mencionada patente fue concedida.

Por lo tanto, como no es posible comprimir todo, los algoritmos de compresión deben asumir que existe algún tipo de sesgo en los mensajes de entrada, es decir, que algunos mensajes son más probables que otros. Esto equivale a asumir una distribución de probabilidad no uniforme sobre los posibles mensajes. La mayoría de los algoritmos de compresión actuales basan este sesgo en la estructura del mensaje: por ejemplo, suponiendo que los caracteres repetidos son más frecuentes que los aleatorios, o que en las imágenes típicas aparecen zonas blancas amplias. En este sentido, la compresión está profundamente relacionada con la probabilidad.

Al estudiar los algoritmos de compresión, es útil distinguir entre dos componentes: el modelo y el codificador. El modelo intenta capturar la distribución de probabilidad de los mensajes mediante el conocimiento o el descubrimiento de su estructura. El codificador aprovecha estas probabilidades para generar códigos más eficientes, asignando códigos más cortos a los mensajes más probables y códigos más largos a los menos probables. Por ejemplo, un modelo podría “entender” que ciertos rostros humanos son más comunes que otros objetos (como una tetera), y el codificador podría enviar mensajes más cortos para formas que parezcan rostros. Este enfoque sería útil en videoconferencias. Sin embargo, en la práctica, los modelos de compresión no suelen ser tan sofisticados y se limitan a patrones comunes como secuencias repetidas. Aunque existen múltiples niveles de complejidad para el modelo, los codificadores tienden a ser bastante genéricos, y en la mayoría de los algoritmos actuales se basan casi exclusivamente en codificación de Huffman o codificación aritmética. En todo caso, la frontera entre el modelo y el codificador no siempre es nítida.

La teoría de la información es la que da cohesión entre el modelo y el codificador. Esta teoría establece una relación clara entre las probabilidades, el contenido de información y la longitud de los códigos. Como veremos, sus predicciones coinciden sorprendentemente bien con la práctica, logrando longitudes de código muy cercanas a los valores teóricos óptimos.

Otra cuestión relevante es cómo evaluar la calidad de un algoritmo de compresión frente a otro. En compresión sin pérdida, algunos criterios básicos incluyen el tiempo necesario para comprimir, el tiempo para descomprimir, el tamaño de los mensajes comprimidos y su aplicabilidad a diferentes tipos de contenido (por ejemplo, si solo funciona bien con Shakespeare o también con Byron). En compresión con pérdida, se añade la dificultad de medir la calidad de la reconstrucción. Suele haber una compensación entre el nivel de compresión, el tiempo de ejecución y la calidad. Según la aplicación concreta, se puede priorizar uno u otro aspecto, y la elección del algoritmo debe hacerse en consecuencia.

Una de las comparaciones más completas entre algoritmos de compresión sin pérdida es el *Archive Comparison Test (ACT)*, realizado por Jeff Gilchrist. Este test evalúa cientos de algoritmos sobre múltiples bases de datos, reportando tiempos y tasas de compresión, y asignando una puntuación basada en un promedio ponderado de ambos.

Teoría de la Información

Entropía

Shannon tomó prestado el concepto de entropía de la física estadística, donde la entropía representa el desorden o la aleatoriedad de un sistema. En dicho contexto, se asume que un sistema puede encontrarse en un conjunto de posibles estados, y en un momento dado existe una distribución de probabilidad sobre esos estados. La entropía se define entonces como:

H(S)=∑s∈Sp(s)log⁡2(1p(s))H(S) = \sum_{s \in S} p(s) \log_2 \left(\frac{1}{p(s)} \right)H(S)=s∈S∑ p(s)log2(p(s)1)

donde SSS es el conjunto de estados posibles y p(s)p(s)p(s) es la probabilidad de que el sistema esté en el estado s∈Ss \in Ss∈S.

Esta fórmula indica que cuanto más uniforme es la distribución de probabilidades, mayor es la entropía (más desorden); y cuanto más sesgada, menor es la entropía. Por ejemplo, si sabemos exactamente en qué estado se encuentra el sistema, la entropía es cero. Esta idea está en línea con la segunda ley de la termodinámica, que establece que la entropía de un sistema cerrado solo puede aumentar.

En el contexto de la teoría de la información, Shannon reemplazó el término "estado" por "mensaje". Así, SSS pasa a ser el conjunto de mensajes posibles, y p(s)p(s)p(s) la probabilidad de cada mensaje. También definió el concepto de "información propia" de un mensaje como:

i(s)=log⁡2(1p(s))i(s) = \log_2 \left(\frac{1}{p(s)} \right)i(s)=log2(p(s)1)

Esta cantidad representa el número de bits de información que contiene el mensaje y, en términos generales, cuántos bits se deberían usar para codificarlo. Según esta definición, los mensajes con mayor probabilidad contienen menos información (por ejemplo, que mañana hará sol en Los Ángeles es menos informativo que decir que va a nevar).

La entropía se define entonces como el promedio ponderado por probabilidad de la información propia de cada mensaje. Representa el número promedio de bits necesarios para codificar un mensaje seleccionado aleatoriamente de acuerdo con esa distribución. Cuanto mayor es la entropía, mayor es el contenido medio de información. Aunque pueda parecer contradictorio, un conjunto de mensajes más aleatorio (con probabilidades más uniformes) contiene, en promedio, más información.

Ejemplos de cálculo de entropía:

1. p(S)={0.25,0.25,0.25,0.125,0.125}p(S) = \{0.25, 0.25, 0.25, 0.125, 0.125\}p(S)={0.25,0.25,0.25,0.125,0.125}

H=3×0.25×log⁡24+2×0.125×log⁡28=1.5+0.75=2.25H = 3 \times 0.25 \times \log_2 4 + 2 \times 0.125 \times \log_2 8 = 1.5 + 0.75 = 2.25H=3×0.25×log24+2×0.125×log28=1.5+0.75=2.25

2. p(S)={0.5,0.125,0.125,0.125,0.125}p(S) = \{0.5, 0.125, 0.125, 0.125, 0.125\}p(S)={0.5,0.125,0.125,0.125,0.125}

H=0.5×log⁡22+4×0.125×log⁡28=0.5+1.5=2H = 0.5 \times \log_2 2 + 4 \times 0.125 \times \log_2 8 = 0.5 + 1.5 = 2H=0.5×log22+4×0.125×log28=0.5+1.5=2

3. p(S)={0.75,0.0625,0.0625,0.0625,0.0625}p(S) = \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\}p(S)={0.75,0.0625,0.0625,0.0625,0.0625}

H=0.75×log⁡2(43)+4×0.0625×log⁡216=0.3+1=1.3H = 0.75 \times \log_2 \left(\frac{4}{3} \right) + 4 \times 0.0625 \times \log_2 16 = 0.3 + 1 = 1.3H=0.75×log2(34)+4×0.0625×log216=0.3+1=1.3

Como se puede observar, cuanto más desigual es la distribución, menor es la entropía.

Pero, ¿por qué usar el logaritmo de la inversa de la probabilidad como medida de la información de un mensaje? Aunque esta relación se demostrará formalmente más adelante, podemos obtener cierta intuición. Por ejemplo, si tenemos n=2in = 2^in=2i mensajes igualmente probables, entonces la probabilidad de cada uno es 1/n1/n1/n, y codificar cada uno requerirá log⁡2n\log_2 nlog2n bits. Esto coincide exactamente con la información propia, ya que i(Si)=log⁡2(1/pi)=log⁡2ni(S_i) = \log_2(1/p_i) = \log_2 ni(Si)=log2(1/pi)=log2n.

Además, otra propiedad deseable es que la información de dos mensajes independientes se sume. Si los mensajes A y B son independientes, su probabilidad conjunta es p(A)p(B)p(A)p(B)p(A)p(B), y la información que contienen juntos es:

i(AB)=log⁡2(1p(A)p(B))=log⁡2(1p(A))+log⁡2(1p(B))=i(A)+i(B)i(AB) = \log_2 \left(\frac{1}{p(A)p(B)} \right) = \log_2 \left(\frac{1}{p(A)} \right) + \log_2 \left(\frac{1}{p(B)} \right) = i(A) + i(B)i(AB)=log2(p(A)p(B)1)=log2(p(A)1)+log2(p(B)1)=i(A)+i(B)

El logaritmo es la función más simple que cumple con esta propiedad aditiva.

La entropía del idioma inglés

Una pregunta interesante es cuánto contenido informativo tiene el idioma inglés. Esta medida podría utilizarse para establecer un límite superior en la compresión posible de textos en inglés, y también permitir comparaciones con otros idiomas en cuanto a su densidad informativa.

Una manera de medir el contenido de información es mediante el número medio de bits por carácter. Si asumimos probabilidades iguales para todos los caracteres imprimibles (unos 96 en un teclado estándar), entonces cada carácter necesitaría ⌈log⁡296⌉=7\lceil \log_2 96 \rceil = 7⌈log296⌉=7 bits.

Sin embargo, si usamos una distribución realista basada en textos en inglés, la entropía se reduce a unos 4.5 bits por carácter. Utilizando codificación óptima (como la de Huffman), el valor promedio puede subir ligeramente, por ejemplo, a 4.7 bits por carácter.

Aún no se han tenido en cuenta las relaciones entre caracteres cercanos. Si dividimos el texto en bloques de 8 caracteres y calculamos la entropía de esos bloques (basándonos en su frecuencia en un corpus de inglés), obtenemos una entropía de aproximadamente 19 bits por bloque, es decir, 2.4 bits por carácter. Agrupando bloques más largos, se ha estimado que la entropía del inglés se acerca a 1.3 bits por carácter. Esta es una estimación teórica, ya que no existe un corpus lo suficientemente grande para calcularla con precisión.

Este valor de 1.3 bits por carácter representa un límite teórico para la compresión sin pérdida de textos en inglés. En la práctica, los compresores generales (como gzip o BOA) alcanzan entre 2 y 3.7 bits por carácter. Para alcanzar el límite de 1.3 sería necesario un compresor que “entendiera” la gramática inglesa, los modismos, etc.

Entropía condicional y cadenas de Markov

Con frecuencia, la probabilidad de un evento (o mensaje) depende del contexto en que ocurre. Al tener en cuenta el contexto, es posible mejorar las predicciones y, en consecuencia, reducir la entropía. Por ejemplo, en texto el contexto puede ser los caracteres anteriores (como veremos en la sección sobre PPM), o en imágenes, los píxeles vecinos (como ocurre en JBIG).

La probabilidad condicional de un evento eee dado un contexto ccc se denota como p(e|c)p(e|c)p(e|c). La probabilidad total del evento se obtiene sumando sobre todos los contextos:

p(e)=∑c∈Cp(c)·p(e|c)p(e) = \sum_{c \in C} p(c) \cdot p(e|c)p(e)=c∈C∑p(c)·p(e|c)

La información propia condicional es:

i(e|c)=log⁡2(1p(e|c))i(e|c) = \log_2 \left(\frac{1}{p(e|c)} \right)i(e|c)=log2(p(e|c)1)

Esta no necesariamente coincide con la información propia sin contexto. Por ejemplo, decir que va a llover en Los Ángeles puede parecer más informativo si no se sabe que es enero.

La entropía condicional promedio se calcula ponderando sobre los contextos y los mensajes:

H(S|C)=∑c∈Cp(c)∑s∈Sp(s|c)log⁡2(1p(s|c))H(S|C) = \sum_{c \in C} p(c) \sum_{s \in S} p(s|c) \log_2 \left(\frac{1}{p(s|c)} \right)H(S|C)=c∈C∑p(c)s∈S∑p(s|c)log2(p(s|c)1)

Si la distribución de SSS es independiente del contexto CCC, entonces H(S|C)=H(S)H(S|C) = H(S)H(S|C)=H(S); de lo contrario, H(S|C)<H(S)H(S|C) < H(S)H(S|C)<H(S). Es decir, el conocimiento del contexto solo puede reducir la entropía.

Shannon definió originalmente la entropía para fuentes de información. Una fuente genera una secuencia infinita de mensajes XkX_kXk, con k∈Zk \in \mathbb{Z}k∈Z, a partir de un conjunto SSS. Si cada mensaje es independiente de los anteriores, la fuente es independiente e idénticamente distribuida (iid). En este caso, la entropía se denomina entropía de primer orden.

Otro tipo de fuente es una cadena de Markov de orden kkk, donde la probabilidad de cada mensaje depende de los kkk mensajes anteriores. El conjunto de posibles combinaciones anteriores constituye los “estados” del sistema. La entropía de una cadena de Markov se basa en las probabilidades condicionales dadas esas secuencias previas.

Un ejemplo de cadena de Markov de primer orden es un modelo que genera píxeles blancos (w) o negros (b) con ciertas probabilidades condicionales, como:

- p(w|w)=0.99p(w|w) = 0.99p(w|w)=0.99
- p(b|w)=0.01p(b|w) = 0.01p(b|w)=0.01
- p(b|b)=0.7p(b|b) = 0.7p(b|b)=0.7
- p(w|b)=0.3p(w|b) = 0.3p(w|b)=0.3

Calculando las probabilidades estacionarias, se obtiene p(b)=1/31p(b) = 1/31p(b)=1/31 y p(w)=30/31p(w) = 30/31p(w)=30/31, y la entropía condicional se aproxima a 0.107 bits por píxel.

En cambio, la entropía de primer orden es aproximadamente 0.206 bits por píxel, es decir, casi el doble. Esto muestra cómo el contexto (la memoria del sistema) reduce la entropía.

Finalmente, la entropía de una fuente arbitraria puede definirse mediante cadenas de longitud nnn de un alfabeto AAA. La entropía normalizada de orden nnn es:

Hn=1n∑X∈Anp(X)log⁡2(1p(X))H_n = \frac{1}{n} \sum_{X \in A^n} p(X) \log_2 \left(\frac{1}{p(X)} \right)Hn=n1X∈An∑p(X)log2(p(X)1)

La entropía de la fuente se define como el límite cuando n→∞n \to \inftyn→∞:

H=lim⁡n→∞HnH = \lim_{n \to \infty} H_nH=n→∞limHn

En general, calcular la entropía de una fuente compleja es extremadamente difícil, ya que requiere estimar con precisión las probabilidades de secuencias muy largas.

Codificación probabilística

En este capítulo abordamos la parte de los algoritmos de compresión que se ocupa de transformar los mensajes en secuencias de bits: el proceso de codificación. En particular, nos enfocamos en cómo usar las probabilidades de los mensajes, ya sean proporcionadas o estimadas por un modelo, para diseñar códigos eficientes. Esta relación entre probabilidad e información nos lleva directamente a los conceptos fundamentales de la teoría de la información.

Códigos prefijo

Un código es un conjunto de reglas que asigna a cada mensaje una secuencia de bits. Un tipo importante de código es el código prefijo, también conocido como prefijo libre. En este tipo de codificación, ningún código puede ser prefijo de otro. Es decir, si un mensaje tiene la secuencia de bits 101, ningún otro mensaje puede comenzar con 101. Esta propiedad permite que la decodificación sea inmediata y unívoca: una vez que se reconoce un código, se sabe exactamente a qué mensaje pertenece, sin ambigüedad ni necesidad de esperar más bits.

Una manera simple de visualizar un código prefijo es mediante un árbol binario, donde cada hoja representa un mensaje y el camino desde la raíz hasta la hoja define el código de ese mensaje. Por ejemplo, si un mensaje se encuentra en la hoja que se alcanza con la secuencia izquierda-derecha-izquierda (0-1-0), su código será 010.

Este tipo de codificación resulta muy útil porque permite una decodificación eficiente y sin ambigüedades, lo que es esencial en sistemas que deben procesar datos en tiempo real.

Relación con la entropía

Existe una estrecha relación entre la entropía y la longitud esperada de los códigos. Si cada mensaje sss tiene una probabilidad p(s)p(s)p(s), entonces, como vimos antes, su información propia es log⁡2(1/p(s))\log_2(1/p(s))log2(1/p(s)). Idealmente, nos gustaría asignar a cada mensaje un código cuya longitud en bits sea exactamente igual a su información propia. Sin embargo, como las longitudes de los códigos deben ser enteras y cumplir la propiedad de prefijo, no siempre se puede lograr esta correspondencia exacta.

Aun así, se ha demostrado que para cualquier distribución de probabilidad, existe un código prefijo cuya longitud media está muy cerca de la entropía. De hecho, hay un límite inferior fundamental: la longitud promedio de cualquier código prefijo no puede ser menor que la entropía del sistema. Además, hay construcciones (como el algoritmo de Huffman) que producen códigos cuya longitud promedio es menor que la entropía más 1.

En resumen, la entropía establece un límite teórico para la compresión: ningún código puede tener una longitud promedio menor que la entropía del conjunto de mensajes. Y aunque los códigos reales deben tener longitudes enteras, con una buena estrategia de codificación se puede acercar mucho a ese límite.

Codificación de Huffman

El algoritmo de Huffman, desarrollado por David Huffman en 1952, es uno de los métodos más conocidos para construir códigos prefijo óptimos cuando se conoce la distribución de probabilidad de los mensajes.

El algoritmo funciona de la siguiente manera: se comienza con una lista de todos los símbolos y sus probabilidades asociadas. En cada paso, se seleccionan los dos símbolos con menor probabilidad y se combinan en un nuevo símbolo cuya probabilidad es la suma de ambas. Este nuevo símbolo representa una rama interna del árbol, y los símbolos originales se convierten en hijos de ese nodo. El proceso se repite hasta que solo queda un símbolo, que será la raíz del árbol. Luego se asignan 0 y 1 a las ramas del árbol para obtener los códigos finales.

Este procedimiento genera códigos prefijo óptimos: minimiza la longitud media de codificación entre todos los códigos prefijo posibles, siempre que las probabilidades sean potencias negativas de 2. Si las probabilidades no cumplen esa condición, el código de Huffman sigue estando muy cerca del óptimo, aunque no sea exactamente igual a la entropía.

El código de Huffman es ampliamente utilizado en muchos estándares de compresión, incluyendo JPEG, MP3, y otros formatos multimedia.

Combinación de mensajes

Una propiedad interesante del algoritmo de Huffman es su comportamiento cuando se combinan mensajes o símbolos. Si se agrupan pares de mensajes en nuevos símbolos, se puede aplicar Huffman a esta nueva distribución. Esto puede resultar en una mejor aproximación a la entropía, ya que la codificación se realiza sobre bloques en lugar de símbolos individuales. En la práctica, sin embargo, el coste computacional de trabajar con combinaciones de símbolos puede ser elevado, por lo que se emplean técnicas como la codificación por bloques con ventanas móviles.

Códigos de Huffman de mínima varianza

Otra variante del algoritmo de Huffman busca minimizar la varianza en la longitud de los códigos. Aunque dos códigos puedan tener la misma longitud media, uno puede tener códigos de longitud muy dispersa y otro más equilibrada. Reducir la varianza es útil en sistemas donde la consistencia del tiempo de procesamiento es importante, como en sistemas en tiempo real.

Los algoritmos de mínima varianza de Huffman realizan pequeñas modificaciones en el árbol de codificación para que las longitudes de los códigos estén más uniformemente distribuidas, aunque eso implique una ligera desviación del óptimo en la longitud promedio.

Codificación aritmética

La codificación aritmética es otra técnica para generar códigos a partir de distribuciones de probabilidad. A diferencia de los códigos prefijo como Huffman, que asignan a cada símbolo una secuencia de bits, la codificación aritmética representa una secuencia completa de símbolos como un solo número real en el intervalo [0,1)[0,1)[0,1).

El procedimiento consiste en dividir recursivamente el intervalo [0,1)[0,1)[0,1) en subintervalos según las probabilidades de los símbolos. Por ejemplo, si tenemos dos símbolos, A y B, con probabilidades 0.6 y 0.4 respectivamente, entonces el intervalo [0,1)[0,1)[0,1) se divide en [0,0.6)[0, 0.6)[0,0.6) para A y [0.6,1)[0.6, 1)[0.6,1) para B. Al procesar una secuencia de símbolos, se restringe el intervalo original sucesivamente, y el resultado final es un número que pertenece al intervalo correspondiente a esa secuencia.

Este número puede luego convertirse en una secuencia de bits que identifica de manera única el mensaje. A mayor precisión en los cálculos, más larga será la secuencia de bits, pero también más eficiente será la compresión.

La codificación aritmética puede lograr una eficiencia muy cercana al límite teórico de la entropía, superando en ocasiones a Huffman, especialmente cuando las probabilidades no son potencias negativas de 2 o cuando se trabaja con secuencias largas.

Sin embargo, la codificación aritmética también presenta desafíos. Su implementación es más compleja y puede ser más lenta. Además, históricamente ha estado sujeta a restricciones de patentes, lo que ha limitado su uso en algunas aplicaciones comerciales.

Implementación con enteros

En teoría, la codificación aritmética utiliza números reales con precisión infinita, lo cual no es factible en la práctica. Por eso, las implementaciones reales utilizan aritmética entera, simulando el proceso mediante técnicas que trabajan con rangos de enteros y aplican redondeos controlados.

Estas implementaciones deben tener especial cuidado en evitar el desbordamiento y en gestionar el subintervalo activo con la precisión necesaria. También deben atender a problemas como la propagación de errores, ya que una mínima alteración en los bits puede afectar la reconstrucción completa del mensaje.

A pesar de estas complicaciones, las versiones con enteros de la codificación aritmética han demostrado ser muy efectivas y se usan en estándares como JPEG 2000 y JBIG.

Aplicaciones de la codificación probabilística

Hasta ahora hemos visto cómo construir códigos eficientes a partir de distribuciones de probabilidad. Sin embargo, en la práctica, la compresión suele involucrar algo más que simplemente aplicar codificación de Huffman o aritmética a símbolos individuales. Existen transformaciones previas que se aplican sobre los datos para mejorar la distribución de los

símbolos y hacer que la codificación final sea más eficiente. Este capítulo explora algunas de esas transformaciones o modelos, que ayudan a exponer patrones y estructuras en los datos, permitiendo una mejor compresión.

Codificación por longitud de secuencia (Run-Length Coding)

La codificación por longitud de secuencia (RLC, por sus siglas en inglés) es una de las formas más simples y antiguas de compresión. Su idea principal es reemplazar secuencias repetidas de un mismo símbolo por una representación más corta que indique el símbolo y cuántas veces se repite.

Por ejemplo, la secuencia AAAAAA podría codificarse como 6A, lo que representa que la letra A se repite 6 veces. Este método es especialmente efectivo cuando los datos contienen muchas repeticiones consecutivas, como ocurre en imágenes en blanco y negro o en ciertos archivos binarios con estructuras repetitivas.

Sin embargo, cuando los datos no tienen muchas repeticiones, la codificación por longitud de secuencia puede ser ineficiente o incluso perjudicial, ya que podría aumentar el tamaño del archivo si se representa cada símbolo individual con un contador innecesario. Por ello, muchas implementaciones combinan RLC con otros métodos de compresión y la aplican selectivamente solo en ciertas zonas del archivo.

Codificación Move-To-Front (MTF)

La codificación Move-To-Front (mover al frente) es una técnica que transforma la secuencia de símbolos de forma que favorezca la aparición de símbolos de baja magnitud, lo cual es ideal para las codificaciones que asignan códigos más cortos a símbolos más frecuentes (como Huffman o aritmética).

La idea es mantener una lista ordenada de todos los símbolos posibles (por ejemplo, los caracteres ASCII), que inicialmente se encuentra en orden alfabético. Cada vez que se procesa un símbolo, se emite su posición actual en la lista, y luego ese símbolo se mueve al principio de la lista.

Este método es útil porque, en muchos datos reales (por ejemplo, texto), los mismos símbolos tienden a repetirse o reaparecer con frecuencia cercana. Como resultado, muchos símbolos acaban ocupando posiciones bajas en la lista y se representan por números pequeños, que son ideales para codificadores posteriores como Huffman.

MTF se utiliza típicamente como paso intermedio en otros algoritmos más complejos, como Burrows-Wheeler.

Codificación Residual: JPEG-LS

En compresión de imágenes, la codificación residual consiste en predecir el valor de un píxel a partir de sus vecinos y almacenar solo la diferencia (residuo) entre el valor real y el valor predicho. Si la predicción es buena, los residuos serán pequeños y tendrán una distribución más concentrada, lo que facilita su compresión.

JPEG-LS (JPEG Lossless) es un estándar basado en esta idea. Su modelo predice el valor de cada píxel utilizando un predictor sencillo que toma como referencia los píxeles vecinos (superior,

izquierdo y superior izquierdo). A partir de esta predicción, se calcula el residuo y se codifica con técnicas como la codificación Golomb-Rice, que es eficiente para valores pequeños.

Este enfoque resulta especialmente efectivo en imágenes médicas, documentos escaneados y otros entornos donde se requiere compresión sin pérdida de información. JPEG-LS logra tasas de compresión similares o mejores que PNG, con menor complejidad computacional.

Codificación con contexto: JBIG

JBIG (Joint Bi-level Image Experts Group) es un estándar para compresión sin pérdida de imágenes binarias, como las de fax o documentos escaneados en blanco y negro.

Este método utiliza codificación aritmética y aprovecha el contexto local de los píxeles: es decir, la decisión sobre cómo codificar un píxel depende de los valores de los píxeles vecinos ya procesados. Por ejemplo, en una imagen de texto, es muy probable que un píxel negro tenga otros píxeles negros cerca, lo cual permite al modelo hacer una predicción más precisa y reducir la entropía condicional.

JBIG define múltiples modelos de contexto basados en patrones de píxeles cercanos. Estos patrones se usan para seleccionar la probabilidad que se le asignará al próximo píxel, mejorando así la eficiencia de la codificación aritmética.

Este enfoque permite una compresión notablemente superior a los métodos que tratan cada píxel de forma independiente.

Codificación con contexto: PPM

PPM (Prediction by Partial Matching) es un modelo probabilístico avanzado para compresión de texto y datos secuenciales. Se basa en construir un modelo estadístico que predice el siguiente símbolo en función del contexto: es decir, una secuencia de símbolos anteriores.

Por ejemplo, si el contexto actual es “th”, PPM puede asignar una alta probabilidad a que el siguiente carácter sea “e”, ya que la palabra “the” es común en inglés. Cuanto más largo sea el contexto utilizado, más precisa puede ser la predicción.

PPM mantiene estadísticas de frecuencias para múltiples longitudes de contexto, y si no encuentra una coincidencia para un contexto largo, recurre a uno más corto. Este proceso se conoce como correspondencia parcial.

El modelo generado por PPM se utiliza junto con codificadores aritméticos para lograr tasas de compresión muy cercanas al límite teórico de la entropía del lenguaje. Es uno de los modelos más sofisticados disponibles para la compresión sin pérdida de texto, aunque su coste computacional puede ser elevado.

Los algoritmos de Lempel-Ziv

Los algoritmos de Lempel-Ziv son una familia de métodos de compresión sin pérdida muy populares y ampliamente utilizados en aplicaciones reales. Se basan en la idea de reemplazar fragmentos repetidos de datos por referencias a apariciones anteriores dentro del mismo texto. La gran ventaja de estos algoritmos es que no requieren conocer previamente la distribución de probabilidad de los símbolos: en lugar de modelar las frecuencias, construyen su modelo sobre

la marcha, mientras leen la entrada. Esto los convierte en algoritmos adaptativos y muy eficientes.

A continuación, exploramos dos variantes principales de esta familia: Lempel-Ziv 77 (LZ77) y Lempel-Ziv-Welch (LZW).

Lempel-Ziv 77 (Ventanas deslizantes)

El algoritmo LZ77, propuesto en 1977 por Abraham Lempel y Jacob Ziv, se basa en el uso de una ventana deslizante sobre los datos de entrada. Esta ventana contiene una parte del texto que ya ha sido codificada, y sirve como diccionario dinámico desde el cual se buscan repeticiones.

La idea es sencilla: en lugar de codificar directamente los caracteres, se busca la coincidencia más larga entre la cadena actual y algún fragmento que ya se encuentre en la ventana. Si se encuentra una coincidencia, se codifica como un triplete:

- la distancia hacia atrás desde la posición actual hasta donde se encuentra la coincidencia en la ventana,
- la longitud de la coincidencia,
- el siguiente carácter que no coincide.

Por ejemplo, si en la posición actual aparece la cadena “ABABABAC”, y ya hemos visto “ABABABA” en la ventana anterior, se puede codificar el fragmento repetido con una distancia y una longitud, y luego indicar que el siguiente carácter nuevo es “C”.

Este método es especialmente efectivo cuando los datos contienen patrones repetitivos, como en textos, código fuente o ciertas imágenes.

La codificación con LZ77 tiene la ventaja de ser completamente automática y no requiere tablas o modelos complejos. Sin embargo, una posible desventaja es que puede resultar ineficiente si no hay suficientes repeticiones o si la ventana es demasiado pequeña para detectar patrones útiles.

Formatos muy conocidos como ZIP, GZIP y PNG utilizan variantes del algoritmo LZ77 como base para la compresión.

Lempel-Ziv-Welch (LZW)

El algoritmo LZW, desarrollado por Terry Welch en 1984 a partir de una idea anterior de Lempel y Ziv (1978), utiliza un enfoque diferente al de las ventanas deslizantes. En lugar de buscar coincidencias en una ventana de texto anterior, LZW construye un diccionario explícito de secuencias de símbolos a medida que procesa la entrada.

El proceso comienza con un diccionario que contiene todos los símbolos individuales del alfabeto de entrada (por ejemplo, los caracteres ASCII). A medida que se leen los datos, se agrupan símbolos en cadenas, y si una nueva cadena no está en el diccionario, se añade. Cada cadena del diccionario se representa con un número (índice), y el algoritmo emite esos índices en lugar de los símbolos originales.

Este método tiene varias ventajas:

- Los índices que representan cadenas más largas permiten una compresión efectiva sin necesidad de codificar caracteres individuales repetidamente.
- El diccionario se construye dinámicamente, adaptándose a la estructura de los datos.
- La codificación y decodificación son eficientes y pueden implementarse sin necesidad de transmitir el diccionario, ya que tanto el codificador como el decodificador lo construyen de forma sincronizada.

LZW es especialmente conocido por ser la base de la compresión en formatos como GIF, TIFF, y el antiguo comando compress en sistemas Unix.

Una limitación de LZW es que, con el tiempo, el diccionario puede crecer demasiado, lo cual afecta la eficiencia. Para mitigar esto, muchas implementaciones imponen un tamaño máximo al diccionario y reinician el proceso una vez alcanzado ese límite.

Otros métodos de compresión sin pérdida

Aparte de los métodos clásicos como Huffman, aritmética y Lempel-Ziv, existen otros algoritmos muy potentes para la compresión sin pérdida. Uno de los más influyentes y eficaces es el algoritmo de Burrows-Wheeler, que no es un compresor por sí mismo, sino una transformación que reordena los datos de entrada para que puedan ser comprimidos más eficientemente por técnicas como Move-To-Front, Run-Length o codificación de Huffman.

Burrows-Wheeler

El algoritmo de Burrows-Wheeler Transform (BWT) fue presentado en 1995 por Michael Burrows y David Wheeler. Su principal innovación es una transformación que reorganiza una cadena de texto para agrupar símbolos similares y así aumentar la redundancia local. Esta transformación no comprime por sí sola, pero prepara los datos para que los algoritmos de compresión puedan funcionar de forma mucho más efectiva.

El funcionamiento de BWT se puede resumir en los siguientes pasos:

1. Construcción de rotaciones: Se generan todas las rotaciones circulares posibles de la cadena de entrada. Por ejemplo, si el texto es BANANA$, una de las rotaciones será ANANA$B, otra NANA$BA, y así sucesivamente. (El símbolo $ se añade al final para marcar el final del texto y asegurar que todas las rotaciones son distintas.)
2. Ordenación lexicográfica: Todas las rotaciones generadas se ordenan en orden alfabético.
3. Construcción de la salida: Se toma la última columna de la matriz ordenada. Esa columna constituye la transformación de Burrows-Wheeler del texto original.

El resultado de esta transformación suele tener largas secuencias de caracteres repetidos, como AAANNNB, lo que facilita muchísimo la compresión posterior mediante Run-Length Coding o codificación de Huffman.

Una de las características más interesantes del BWT es que es completamente reversible: a partir de la cadena transformada (y la posición del símbolo original en la tabla), se puede reconstruir exactamente el texto original. Esta reversibilidad lo convierte en una herramienta ideal para compresión sin pérdida.

BWT es la base de varios compresores modernos, como bzip2, que combina esta transformación con codificación Move-To-Front y Huffman para lograr tasas de compresión superiores a las de GZIP.

Técnicas de compresión con pérdida

Mientras que los métodos de compresión sin pérdida buscan conservar íntegramente los datos originales, las técnicas de compresión con pérdida permiten reducir aún más el tamaño de los datos sacrificando parte de la información, siempre que esta pérdida no afecte significativamente a la calidad percibida por el usuario. Esto es especialmente útil en imágenes, audio y vídeo, donde el sistema perceptual humano es más tolerante a ciertas imperfecciones.

En este capítulo se abordan tres de las técnicas fundamentales utilizadas en la compresión con pérdida: cuantificación escalar, cuantificación vectorial y codificación por transformada.

Cuantificación escalar

La cuantificación escalar es la forma más básica de compresión con pérdida. Consiste en tomar valores individuales de una señal (como muestras de audio o valores de píxeles) y redondearlos al valor más cercano dentro de un conjunto finito de valores predefinidos. Este proceso reduce la precisión de los datos, pero también el número de bits necesarios para representarlos.

Por ejemplo, si una imagen tiene píxeles representados con valores entre 0 y 255, una cuantificación escalar podría reducir el número de niveles a 16, mapeando así varios valores originales a un mismo valor de salida. Esta reducción provoca pérdida de información, pero si se elige bien el número de niveles y los valores de cuantificación, el impacto visual puede ser mínimo.

Este tipo de cuantificación es utilizada ampliamente como paso intermedio en compresores más complejos, y es muy fácil de implementar.

Cuantificación vectorial

La cuantificación vectorial extiende la idea de la cuantificación escalar al operar con grupos de valores (vectores) en lugar de elementos individuales. Por ejemplo, en lugar de procesar cada píxel de una imagen por separado, se puede trabajar con bloques de varios píxeles, o con ventanas de muestras en una señal de audio.

En este método, se define un diccionario de vectores representativos. Cada vector de entrada se reemplaza por el índice del vector más cercano en ese diccionario. Esto permite representar patrones complejos y capturar dependencias entre elementos cercanos, lo que suele dar lugar a una mejor calidad de compresión en comparación con la cuantificación escalar.

La ventaja principal de la cuantificación vectorial es que puede adaptarse a las correlaciones entre los datos y explotar estructuras repetitivas. Sin embargo, construir y buscar en el

diccionario puede ser computacionalmente costoso, por lo que se requieren algoritmos eficientes para aplicarla en tiempo real o en archivos grandes.

Codificación por transformada

La codificación por transformada es una de las técnicas más utilizadas en compresión con pérdida, especialmente en imágenes y vídeo. Su objetivo es transformar los datos originales (por ejemplo, valores de píxeles) a un nuevo dominio donde la información útil esté concentrada en pocos valores, y donde las componentes menos importantes puedan eliminarse sin que el usuario lo perciba.

Un ejemplo clásico es la Transformada Discreta del Coseno (DCT), que se aplica a bloques de imágenes en el estándar JPEG. Esta transformada convierte una señal espacial (como la intensidad de píxeles) en una combinación de frecuencias. Como el ojo humano es menos sensible a los detalles de alta frecuencia, estas componentes se pueden eliminar o comprimir más agresivamente sin afectar mucho la percepción visual.

El proceso general consiste en:

1. Dividir la señal (imagen, audio, etc.) en bloques.
2. Aplicar una transformada (como la DCT) a cada bloque.
3. Cuantificar los coeficientes resultantes, eliminando o reduciendo los menos importantes.
4. Codificar los coeficientes con técnicas como Run-Length y Huffman.

Este enfoque permite una compresión muy eficiente con una pérdida controlada de calidad. Además de JPEG, otras tecnologías como MPEG o AAC también se basan en la codificación por transformada, aunque utilizan diferentes tipos de transformadas (por ejemplo, la Transformada Discreta de Fourier o la Transformada Wavelet).

Estudio de caso — JPEG y MPEG

Hasta ahora hemos explorado diversas técnicas de compresión con y sin pérdida. En este capítulo se ilustran estas técnicas a través de dos de los estándares más influyentes y ampliamente utilizados: JPEG para la compresión de imágenes fijas y MPEG para la compresión de vídeo. Ambos estándares emplean muchas de las ideas teóricas desarrolladas previamente, como la codificación por transformada, la cuantificación y la codificación entropía.

JPEG

JPEG (Joint Photographic Experts Group) es un estándar para la compresión de imágenes fotográficas, principalmente con pérdida. Es especialmente eficiente para imágenes naturales, como fotografías, donde las pequeñas pérdidas de precisión son prácticamente imperceptibles.

El flujo de trabajo del algoritmo JPEG se puede dividir en varias etapas:

1. Conversión de color (opcional):
   En muchos casos, la imagen se convierte de RGB a un espacio de color más eficiente para la compresión, como YCbCr, donde se separan la luminancia (Y) y las dos componentes

de crominancia (Cb y Cr). Esto se debe a que el ojo humano es más sensible a la luminancia que al color, lo que permite comprimir más agresivamente las componentes cromáticas.

2. Submuestreo cromático (opcional):
   En el espacio YCbCr, se pueden reducir (submuestrear) las componentes de color sin afectar demasiado la percepción. Por ejemplo, se puede tomar una muestra de Cb y Cr cada 2×2 píxeles en lugar de cada píxel, reduciendo así el tamaño de los datos.

3. División en bloques de 8×8 píxeles:
   La imagen se divide en bloques de 8 por 8 píxeles, y cada uno se procesa por separado.

4. Aplicación de la Transformada Discreta del Coseno (DCT):
   A cada bloque se le aplica la DCT, que convierte los valores espaciales del bloque (intensidades) en una representación en el dominio de la frecuencia. Los primeros coeficientes representan las variaciones más suaves (baja frecuencia), y los últimos corresponden a los detalles finos (alta frecuencia).

5. Cuantificación:
   Los coeficientes DCT se cuantifican dividiendo cada uno por un valor de una tabla de cuantificación y redondeando el resultado. Esto es donde ocurre la compresión con pérdida, ya que se eliminan o simplifican los componentes menos importantes. Esta etapa puede ajustarse con un parámetro de calidad.

6. Reordenamiento en zigzag y codificación por longitud de secuencia:
   Los coeficientes cuantificados se ordenan en un patrón zigzag para agrupar los ceros consecutivos, lo que favorece la siguiente etapa.

7. Codificación de Huffman (u otra codificación entropía):
   Finalmente, los datos ordenados se codifican usando Huffman u otro método de codificación entropía.

La decodificación simplemente invierte todos estos pasos. Aunque parte de la información original se pierde, la calidad visual suele mantenerse muy alta con ratios de compresión bastante elevados.

JPEG también define un modo sin pérdida, aunque es mucho menos utilizado.

MPEG

MPEG (Moving Picture Experts Group) es una familia de estándares para la compresión de vídeo digital. Al igual que JPEG, se basa en la codificación por transformada y cuantificación, pero incorpora una etapa adicional esencial: la compresión temporal, que explota la redundancia entre cuadros sucesivos de un vídeo.

El principio básico es que en un vídeo muchas de las imágenes consecutivas son muy parecidas. En lugar de almacenar cada una completa, MPEG almacena solo algunas imágenes completas (llamadas I-frames o cuadros intra), y el resto se codifica como diferencias respecto a cuadros anteriores (P-frames) o posteriores (B-frames).

Las etapas principales de compresión en MPEG son:

1. División del vídeo en secuencias de cuadros (frames):
   Las secuencias se dividen en grupos de imágenes (GOP: Group of Pictures), que combinan I, P y B frames.

2. Predicción de movimiento:
   Para los P y B frames, se calcula el movimiento de bloques entre cuadros. Por ejemplo, si un objeto se mueve ligeramente entre dos cuadros, se puede codificar simplemente como "el mismo bloque, desplazado". Esto se conoce como compensación de movimiento.

3. Cálculo del residuo:
   Se calcula la diferencia entre el bloque real y el predicho, y ese residuo se transforma con DCT y se cuantifica, igual que en JPEG.

4. Codificación entropía:
   Los residuos y los vectores de movimiento se codifican mediante Huffman u otros métodos.

Gracias a estas técnicas, MPEG logra tasas de compresión mucho más altas que JPEG al explotar no solo la redundancia espacial dentro de cada imagen, sino también la redundancia temporal entre imágenes.

Existen varios estándares MPEG, entre ellos:

- MPEG-1: usado en VCD y MP3.
- MPEG-2: utilizado en DVD y transmisión de televisión digital (como DVB).
- MPEG-4: diseñado para web y multimedia interactiva.
- H.264 / MPEG-4 AVC: estándar moderno de alta compresión y calidad.

Cada uno introduce mejoras técnicas, pero todos comparten los mismos principios fundamentales de predicción, transformada y codificación entropía.

Otros métodos de codificación con transformadas con pérdida

Además de las transformadas clásicas como la DCT empleada en JPEG o MPEG, existen otros enfoques de compresión con pérdida que se basan en transformaciones alternativas. Estas técnicas ofrecen ventajas particulares en términos de calidad visual, adaptabilidad o eficiencia para ciertos tipos de datos. En este capítulo se describen tres de ellas: compresión por wavelets, compresión fractal y compresión basada en modelos.

Compresión por wavelets

La compresión por wavelets es una técnica más moderna que ha ganado popularidad como alternativa a la DCT, especialmente en estándares como JPEG 2000. Mientras que la DCT analiza las frecuencias de un bloque fijo de píxeles, los wavelets permiten representar una señal a múltiples escalas y resoluciones al mismo tiempo.

La transformada wavelet descompone una imagen (o señal) en diferentes niveles de detalle, dividiendo la información en una parte de baja frecuencia (la aproximación) y varias de alta frecuencia (los detalles horizontales, verticales y diagonales). Esta jerarquía multiescala permite eliminar información menos relevante en zonas más detalladas, sin afectar tanto la calidad visual.

Una de las principales ventajas de las wavelets frente a la DCT es que no requieren dividir la imagen en bloques. Esto evita los artefactos de bloque (bloqueo) típicos de JPEG, y permite una compresión más suave y progresiva. Además, los wavelets ofrecen soporte nativo para escalabilidad (por ejemplo, visualizar primero una versión de baja resolución e ir refinando) y para compresión sin pérdida.

Aunque su implementación es más compleja, la compresión por wavelets representa un avance importante en la calidad y flexibilidad de los sistemas de compresión con pérdida.

Compresión fractal

La compresión fractal es una técnica no convencional que se basa en la autosemejanza de las imágenes: la idea de que ciertas partes de una imagen se parecen a otras a diferentes escalas. Esta propiedad es especialmente común en imágenes naturales como montañas, nubes, árboles o texturas.

El proceso de compresión fractal consiste en dividir la imagen en bloques pequeños y, para cada bloque, buscar otra región (posiblemente más grande y transformada) que se le parezca. En lugar de almacenar directamente los píxeles, se almacena la transformación matemática necesaria para reconstruir el bloque a partir de la región encontrada. Estas transformaciones incluyen rotaciones, escalados y ajustes de brillo/contraste.

Durante la decodificación, el sistema aplica las transformaciones recursivamente para aproximar la imagen original. Aunque esta reconstrucción suele tomar varias iteraciones, el resultado converge hacia una imagen similar a la original.

La compresión fractal ofrece algunas ventajas interesantes:

- Alta relación de compresión en imágenes con mucha autosemejanza.
- Posibilidad de rescalado sin pérdida aparente de calidad, ya que las transformaciones pueden aplicarse a cualquier resolución.
- Modelos matemáticos compactos en lugar de datos puros.

Sin embargo, tiene desventajas importantes: la compresión es muy costosa computacionalmente, y no siempre se obtienen buenos resultados con imágenes muy detalladas o ruidosas. Por estas razones, la compresión fractal no se ha adoptado ampliamente en estándares comerciales, aunque sigue siendo objeto de investigación.

Compresión basada en modelos

La compresión basada en modelos es una aproximación que busca representar los datos utilizando una representación semántica o estructurada del contenido en lugar de almacenar los

datos brutos o transformados. Este enfoque es particularmente potente en aplicaciones específicas, como gráficos 3D, imágenes médicas o secuencias de vídeo sintéticas.

En lugar de comprimir píxel por píxel, la idea es identificar y describir estructuras significativas como formas geométricas, movimientos, texturas o incluso objetos completos. Por ejemplo, en una animación de un rostro humano, no es necesario codificar cada imagen del vídeo: basta con codificar un modelo 3D del rostro y los parámetros de su movimiento y expresión.

Este tipo de compresión ofrece un enorme potencial de reducción de datos, ya que puede representar escenas complejas con muy poca información. Además, permite una manipulación posterior más sencilla (por ejemplo, cambiar ángulos de cámara, reiluminar escenas, etc.).

El principal desafío de la compresión basada en modelos es que requiere conocimiento específico del dominio y algoritmos avanzados de análisis de escena. No es adecuada para datos arbitrarios, pero en contextos bien definidos puede superar ampliamente a los métodos tradicionales.