WIKIPEDIA

Shannon–Fano coding

In the field of data compression, **Shannon–Fano coding**, named after Claude Shannon and Robert Fano, is a name given to two different but related techniques for constructing a prefix code based on a set of symbols and their probabilities (estimated or measured).

- **Shannon's method** chooses a prefix code where a source symbol i is given the codeword length $l_i = \lceil -\log_2 p_i \rceil$. One common way of choosing the codewords uses the binary expansion of the cumulative probabilities. This method was proposed in Shannon's "A Mathematical Theory of Communication" (1948), his article introducing the field of information theory.
- **Fano's method** divides the source symbols into two sets ("0" and "1") with probabilities as close to 1/2 as possible. Then those sets are themselves divided in two, and so on, until each set contains only one symbol. The codeword for that symbol is the string of "0"s and "1"s that records which half of the divides it fell on. This method was proposed in a later technical report by Fano (1949).

Shannon–Fano codes are suboptimal in the sense that they do not always achieve the lowest possible expected codeword length, as Huffman coding does.[1] However, Shannon–Fano codes have an expected codeword length within 1 bit of optimal. Fano's method usually produces encoding with shorter expected lengths than Shannon's method. However, Shannon's method is easier to analyse theoretically.

Shannon–Fano coding should not be confused with Shannon–Fano–Elias coding (also known as Elias coding), the precursor to arithmetic coding.

Contents

Naming

Regarding the confusion in the two different codes being referred to by the same name, Krajči et al[2] write:

> Around 1948, both Claude E. Shannon (1948) and Robert M. Fano (1949) independently proposed two different source coding algorithms for an efficient description of a discrete memoryless source. Unfortunately, in spite of being different, both schemes became known under the same name *Shannon–Fano coding*.
>
> There are several reasons for this mixup. For one thing, in the discussion of his coding scheme, Shannon mentions Fano's scheme and calls it "substantially the same" (Shannon, 1948, p. 17). For another, both Shannon's and Fano's coding schemes are similar in the sense that they both are efficient, but *suboptimal* prefix-free coding schemes with a similar performance

Shannon's (1948) method, using predefined word lengths, is called **Shannon–Fano coding** by Cover and Thomas[3], Goldie and Pinch[4], Jones and Jones[5], and Han and Kobayashi.[6] It is called **Shannon coding** by Yeung.[7]

Fano's (1949) method, using binary division of probabilities, is called **Shannon–Fano coding** by Salomon[8] and Gupta.[9] It is called **Fano coding** by Krajči et al[2].

Shannon's code: predefined word lengths

Shannon's algorithm

Shannon's method starts by deciding on the lengths of all the codewords, then picks a prefix code with those word lengths.

Given a source with probabilities $p_1, p_2, \ldots, p_n$ the desired codeword lengths are $l_i = \lceil -\log_2 p_i \rceil$. Here, $\lceil x \rceil$ is the ceiling function, meaning the smallest integer greater than or equal to x.

Once the codeword lengths have been determined, we must choose the codewords themselves. One method is to pick codewords in order from most probable to least probable symbols, picking each codeword to be the lexicographically first word of the correct length that maintains the prefix-free property.

A second method makes use of cumulative probabilities. First, the probabilities are written in decreasing order $p_1 \geq p_2 \geq \cdots \geq p_n$. Then, the cumulative probabilities are defined as

$$c_1 = 0, \qquad c_i = \sum_{j=1}^{i-1} p_i \text{ for } i \geq 2,$$

so $c_1 = 0, c_2 = p_1, c_3 = p_1 + p_2$ and so on. The codeword for symbol i is chosen to be the first l_i binary digits in the binary expansion of c_i.

Example

This example shows the construction of a Shannon–Fano code for a small alphabet. There 5 different source symbols. Suppose 39 total symbols have been observed with the following frequencies, from which we can estimate the symbol probabilities.

Symbol	A	B	C	D	E
Count	15	7	6	6	5
Probabilities	0.385	0.179	0.154	0.154	0.128

This source has entropy $H(X) = 2.186$ bits.

For the Shannon–Fano code, we need to calculate the desired word lengths $l_i = \lceil -\log_2 p_i \rceil$.

Symbol	A	B	C	D	E
Probabilities	0.385	0.179	0.154	0.154	0.128
$-\log_2 p_i$	1.379	2.480	2.700	2.700	2.963
Word lengths $\lceil -\log_2 p_i \rceil$	2	3	3	3	3

We can pick codewords in order, choosing the lexicographically first word of the correct length that maintains the prefix-free property. Clearly A gets the codeword 00. To maintain the prefix-free property, B's codeword may not start 00, so the lexicographically first available word of length 3 is 010. Continuing like this, we get the following code:

Symbol	A	B	C	D	E
Probabilities	0.385	0.179	0.154	0.154	0.128
Word lengths $\lceil -\log_2 p_i \rceil$	2	3	3	3	3
Codewords	00	010	011	100	101

Alternatively, we can use the cumulative probability method.

Symbol	A	B	C	D	E
Probabilities	0.385	0.179	0.154	0.154	0.128
Cumulative probabilities	0.000	0.385	0.564	0.718	0.872
...in binary	0.00000	0.01100	0.10010	0.10110	0.11011
Word lengths $\lceil -\log_2 p_i \rceil$	2	3	3	3	3
Codewords	00	011	100	101	110

Note that although the codewords under the two methods are different, the word lengths are the same. We have lengths of 2 bits for A, and 3 bits for B, C, D and E, giving an average length of

$$\frac{2\,\text{bits} \cdot (15) + 3\,\text{bits} \cdot (7+6+6+5)}{39\,\text{symbols}} \approx 2.62\,\text{bits per symbol},$$

which is within one bit of the entropy.

Expected word length

For Shannon's method, the word lengths satisfy

$$l_i = \lceil -\log_2 p_i \rceil \leq -\log_2 p_i + 1.$$

Hence the expected word length satisfies

$$\mathbb{E}L = \sum_{i=1}^{n} p_i l_i \leq \sum_{i=1}^{n} p_i(-\log_2 p_i + 1) = -\sum_{i=1}^{n} p_i \log_2 p_i + \sum_{i=1}^{n} p_i = H(X) + 1.$$

Here, $H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$ is the entropy, and Shannon's source coding theorem says that any code must have an average length of at least $H(X)$. Hence we see that the Shannon–Fano code is always within one bit of the optimal expected word length.

Fano's code: binary splitting

Outline of Fano's code

In Fano's method, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1". As long as any sets with more than one member remain, the same process is repeated on those sets, to determine successive digits of their codes. When a set has been reduced to one symbol this means the symbol's code is complete and will not form the prefix of any other symbol's code.

The algorithm produces fairly efficient variable-length encodings; when the two smaller sets produced by a partitioning are in fact of equal probability, the one bit of information used to distinguish them is used most efficiently. Unfortunately, Shannon–Fano coding does not always produce optimal prefix codes; the set of probabilities {0.35, 0.17, 0.17, 0.16, 0.15} is an example of one that will be assigned non-optimal codes by Shannon–Fano coding.

Fano's version of Shannon–Fano coding is used in the `IMPLODE` compression method, which is part of the `ZIP` file format.[10]

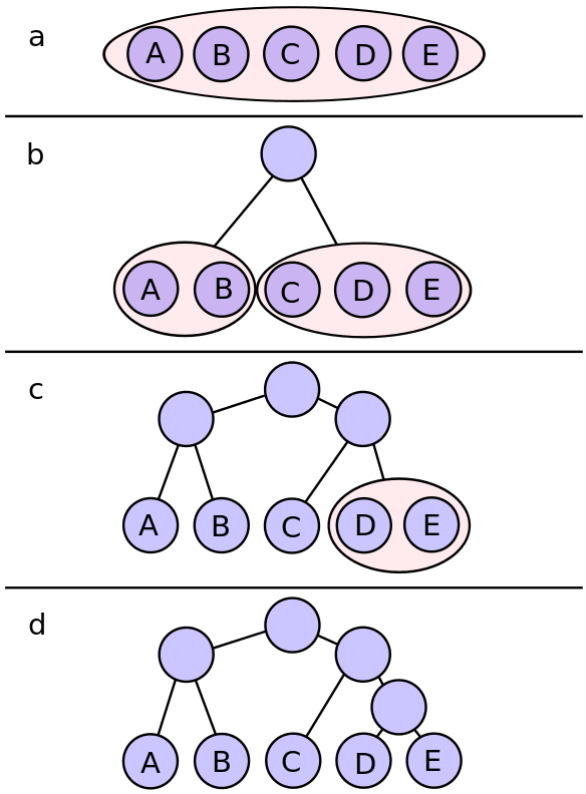
The Shannon–Fano tree

A Shannon–Fano tree is built according to a specification designed to define an effective code table. The actual algorithm is simple:

1. For a given list of symbols, develop a corresponding list of probabilities or frequency counts so that each symbol's relative frequency of occurrence is known.
2. Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the left and the least common at the right.
3. Divide the list into two parts, with the total frequency counts of the left part being as close to the total of the right as possible.
4. The left part of the list is assigned the binary digit 0, and the right part is assigned the digit 1. This means that the codes for the symbols in the first part will all start with 0, and the codes in the second part will all start with 1.
5. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

Example

We continue with the previous example.



Shannon–Fano Algorithm

Symbol	A	B	C	D	E
Count	15	7	6	6	5
Probabilities	0.385	0.179	0.154	0.154	0.128

All symbols are sorted by frequency, from left to right (shown in Figure a). Putting the dividing line between symbols B and C results in a total of 22 in the left group and a total of 17 in the right group. This minimizes the difference in totals between the two groups.

With this division, A and B will each have a code that starts with a 0 bit, and the C, D, and E codes will all start with a 1, as shown in Figure b. Subsequently, the left half of the tree gets a new division between A and B, which puts A on a leaf with code 00 and B on a leaf with code 01.

After four division procedures, a tree of codes results. In the final tree, the three symbols with the highest frequencies have all been assigned 2-bit codes, and two symbols with lower counts have 3-bit codes as shown table below:

Symbol	A	B	C	D	E
Probabilities	0.385	0.179	0.154	0.154	0.128
First division	0		1		
Second division	0	1	0	1	
Third division				0	1
Codewords	00	01	10	110	111

This results in lengths of 2 bits for A, B and C and per 3 bits for D and E, giving an average length of

$$\frac{2\,\text{bits} \cdot (15+7+6) + 3\,\text{bits} \cdot (6+5)}{39\,\text{symbols}} \approx 2.28\,\text{bits per symbol.}$$

We see that Fano's method, with an average length of 2.28, has outperformed Shannon's method, with an average length of 2.62.

Expected word length

It is shown by Krajči et al[2] that the expected length of Fano's method has expected length bounded above by $\mathbb{E}L \leq H(X) + 1 - p_{\min}$, where $p_{\min} = \min_i p_i$ is the probability of the least common symbol.

Comparison with other coding methods

Neither Shannon–Fano algorithm is guaranteed to generate an optimal code. For this reason, Shannon–Fano codes are almost never used; Huffman coding is almost as computationally simple and produces prefix codes that always achieve the lowest possible expected code word length, under the constraints that each symbol is represented by a code formed of an integral number of bits. This is a constraint that is often unneeded, since the codes will be packed end-to-end in long sequences. If we consider groups of codes at a time, symbol-by-symbol Huffman coding is only optimal if the probabilities of the symbols are independent and are some power of a half, i.e., $1/2^k$. In most situations, arithmetic coding can produce greater overall compression than either Huffman or Shannon–Fano, since it can encode in fractional numbers of bits which more closely approximate the actual information content of the symbol. However, arithmetic coding has not superseded Huffman the way that Huffman supersedes Shannon–Fano, both because arithmetic coding is more computationally expensive and because it is covered by multiple patents.

Huffman coding

A few years later, David A. Huffman (1949)[11] gave a different algorithm that always produces an optimal tree for any given symbol probabilities. While Fano's Shannon–Fano tree is created by dividing from the root to the leaves, the Huffman algorithm works in the opposite direction, merging from the leaves to the root.

1. Create a leaf node for each symbol and add it to a priority queue, using its frequency of occurrence as the priority.
2. While there is more than one node in the queue:
    1. Remove the two nodes of lowest probability or frequency from the queue
    2. Prepend 0 and 1 respectively to any code already assigned to these nodes
    3. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
    4. Add the new node to the queue.
3. The remaining node is the root node and the tree is complete.

Example with Huffman coding

We use the same frequencies as for the Shannon–Fano example above, viz:

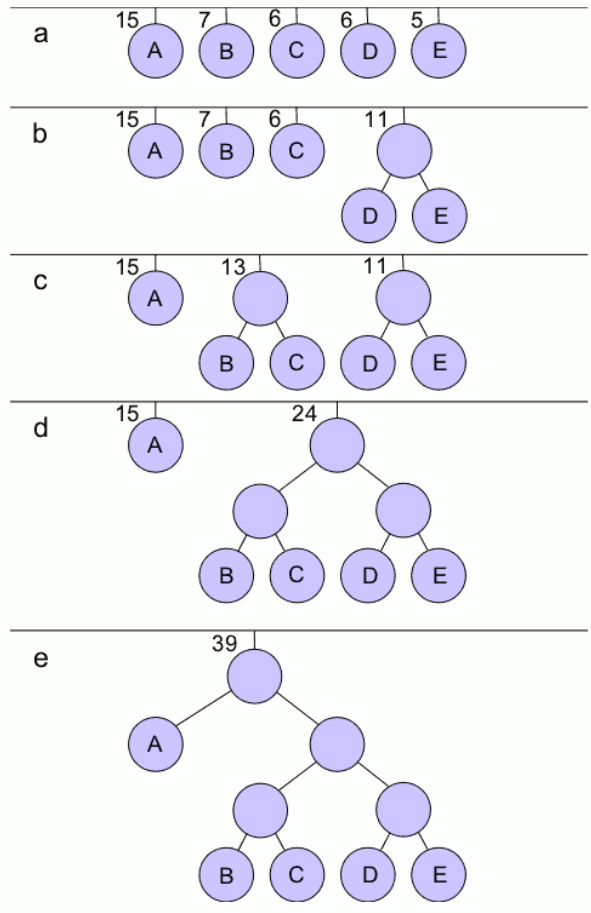
Symbol	A	B	C	D	E
Count	15	7	6	6	5
Probabilities	0.385	0.179	0.154	0.154	0.128

In this case D & E have the lowest frequencies and so are allocated 0 and 1 respectively and grouped together with a combined probability of 0.282. The lowest pair now are B and C so they're allocated 0 and 1 and grouped together with a combined probability of 0.333. This leaves BC and DE now with the lowest probabilities so 0 and 1 are prepended to their codes and they are combined. This then leaves just A and BCDE, which have 0 and 1 prepended respectively and are then combined. This leaves us with a single node and our algorithm is complete.

The code lengths for the different characters this time are 1 bit for A and 3 bits for all other characters.

Symbol	A	B	C	D	E
Codewords	0	100	101	110	111

This results in the lengths of 1 bit for A and per 3 bits for B, C, D and E, giving an average length of



Huffman Algorithm

$$\frac{1\,\text{bit} \cdot 15 + 3\,\text{bits} \cdot (7+6+6+5)}{39\,\text{symbols}} \approx 2.23\,\text{bits per symbol.}$$

We see that the Huffman code has outperformed both types of Shannon–Fano code, which had expected lengths of 2.62 and 2.28.

Notes

1. Kaur, Sandeep; Singh, Sukhjeet (May 2016). "Entropy Coding and Different Coding Techniques" (https://pdfs.semanticscholar.org/4253/7898a836d0384c6689a3c098b823309ab723.pdf) (PDF). *Journal of Network Communications and Emerging Technologies*. **6** (5): 5. Retrieved 3 December 2019.
2. Stanislav Krajči, Chin-Fu Liu, Ladislav Mikeš and Stefan M. Moser (2015), "Performance analysis of Fano coding", *2015 IEEE International Symposium on Information Theory (ISIT)*.
3. Thomas M. Cover and Joy A. Thomas (2006), *Elements of Information Theory* (2nd ed.), Wiley–Interscience. "Historical Notes" to Chapter 5.
4. Charles M. Goldie and Richard G. E. Pinch (1991), *Communication Theory*, Cambridge University Press. Section 1.6.
5. Gareth A. Jones and J. Mary Jones (2012), *Information and Coding Theory* (Springer). Section 3.4.
6. Te Sun Han and Kingo Kobayashi (2007), *Mathematics of Information and Coding*, American Mathematical Society. Subsection 3.7.1.
7. Raymond W Yeung (2002), *A First Course in Information Theory*, Springer. Subsection 3.2.2.
8. David Salomon (2013), *Data Compression: The Complete Reference*, Springer. Section 2.6.

9. Prakash C. Gupta (2006), *Data Communications and Computer Networks*, Phi Publishing. Subsection 1.11.5.
10. "APPNOTE.TXT - .ZIP File Format Specification" (http://www.pkware.com/documents/casestudies/APPNOTE.TXT). PKWARE Inc. 2007-09-28. Retrieved 2008-01-06. "The Imploding algorithm is actually a combination of two distinct algorithms. The first algorithm compresses repeated byte sequences using a sliding dictionary. The second algorithm is used to compress the encoding of the sliding dictionary output, using multiple Shannon–Fano trees."
11. Huffman, D. (1952). "A Method for the Construction of Minimum-Redundancy Codes" (http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf) (PDF). *Proceedings of the IRE*. **40** (9): 1098–1101. doi:10.1109/JRPROC.1952.273898 (https://doi.org/10.1109%2FJRPROC.1952.273898).

References

- Fano, R.M. (1949). "The transmission of information" (https://archive.org/details/fano-tr65.7z). *Technical Report No. 65*. Cambridge (Mass.), USA: Research Laboratory of Electronics at MIT.
- Shannon, C.E. (July 1948). "A Mathematical Theory of Communication" (https://archive.org/details/ost-engineering-shannon1948). *Bell System Technical Journal*. **27**: 379–423.