CS361: Introduction to Computer Security

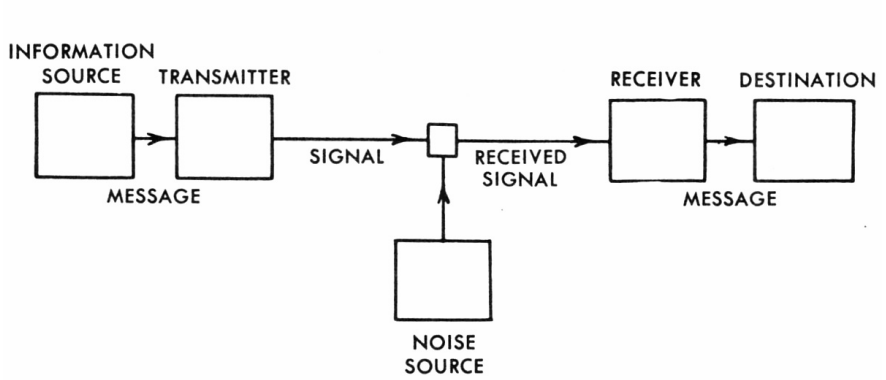
Information Theory

Dr. Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: February 19, 2020 at 15:35

What is Information

Just what is information? The following is the fundamental paradigm of information theory:



Information is any content to be conveyed from the sender to receiver. It is sent in the form of one or more *messages*. Our goal is to quantify this sitution.

Some Information Theory

Information theory is the field of study that quantifies the *content* of messages and the *capacity* of channels on which information may flow. It asks questions such as the following:

1. How much information is encoded in a particular message?
2. How efficiently can a given alphabet/language be transmitted?
3. What is the maximum capacity of a given transmission medium?
4. How is that capacity or efficiency reduced by interference /noise?

These questions all have very precise and deep answers, that are beyond the scope of this course. We'll only scratch the surface.

Why Do We Care?

Information theory is very important in computer science. It affects all communication, hardware design, protocol design, cryptography, fault-tolerance, etc.

For example, in our current context it is useful to know how much information can be transmitted over a specific covert channel. This is the "bandwidth" of the channel.

If the bandwidth is sufficiently small, perhaps we want to ignore it, but if it's large that suggests a potential problem.

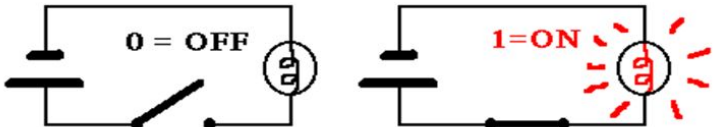
Quantifying Information: Thought Experiment

How much information is contained in each of the following messages? How do you judge?

- A 1-bit binary number.
- An n-bit binary number.
- A single decimal digit.
- A two digit decimal number.
- "The attack will come at dawn."
- "One if by land; two if by sea."
- The King James Bible or the Collected Works of Poe.

Information Content

The information content of a message consists simply of the number of 1's and 0's it takes to transmit it.



Quantifying Information

If you have n bits, you have 2^n possible messages. Why?

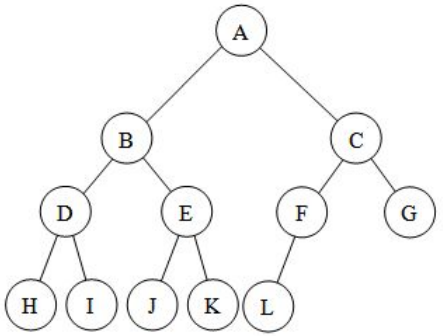
Conversely, if you have 2^n possible messages, then n bits is enough to select unambiguously any single one of those. Why?

Aside: Mathematically, the information content is related to the logarithm (base 2) of the size of the potential message space.

$\log_2(x)$ is the value y such that $x = 2^y$. E.g., $\log_2(16) = 4$ since $16 = 2^4$. Can you see why 4 bits suffices to select one of 16 possible choices. Hint: think about binary trees.

Binary Tree Representation

If we represent all possible *choices* in any decision as the leaves of a "complete" binary tree. How long can any path be?



If there are n nodes, the longest path is bounded by $\log_2 n$.

Information as Uncertainty

It is tempting to think that some messages are more important than others. For example, leaking a decryption key may be more costly than leaking some other message. That's a *semantic* approach to information.

Information theory can't deal with that aspect of security. It can only quantify the *information content* of a message in terms of the space of possible messages that might have been sent or the "uncertainty" on the receiver's side of what message might have been sent. That's basically a *syntactic* approach in which the symbols are *uninterpreted*.

Information as Uncertainty

The information content of a message depends on the shared knowledge of the sender and receiver and the size of the message-space.

The amount of information in a message sent increases with the uncertainty as to what message actually will be sent.

No uncertainty means no information.

Intuitively, the information content is something like the logarithm of the size of the potential message space. E.g., if there are four possible messages, then a message has two bits of information. Explain.

Information as Uncertainty

"The concept in this theory at first seems disappointing and bizarre—disappointing because it has nothing to do with meaning, and bizarre because it deals not with a single message, but with the statistical character of a whole ensemble of messages."
–Warren Weaver

"The word 'information' relates not so much to what you do say, as to what you could say. The mathematical theory of communication deals with the carriers of information, symbols and signals, not with information itself. That is, information is the measure of your freedom of choice when you select a message." –Warren Weaver

Information Content as Bits

Information content is commonly measured in terms of *bits*. "Bit" has two connotations, *which are not the same*:

- the quantity of information required to disambiguate a choice between two distinct possibililities (continuous quantity);
- a binary digit (a discrete quantity).

Information Content as Bits

The *information content* of a message is measured in bits. The *capacity* of a channel is measured in bits per second (bps).

In general, the best way of transmitting (encoding) a message is that way which minimizes the number of bits required, on average. What does that mean?

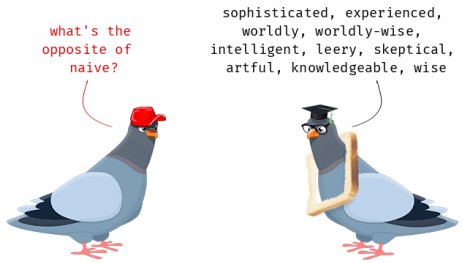
Information Content as Bits

Suppose I tell you that transmitting one out of 16 possible messages $A \ldots P$ requires 4 bits of information be sent. Is this true? Why or why not?

Information Content as Bits

Suppose I tell you that transmitting one out of 16 possible messages $A \ldots P$ requires 4 bits of information be sent. Is this true? Why or why not?

Let's call the code using 4 bits per message *the naïve encoding*—just number the messages and transmit the number as our code.



It's only naïve in the sense that it doesn't require any thought.

Information Content as Bits

Can you do better than 4 bits for sending one message? What would it mean to do better?

Information Content as Bits

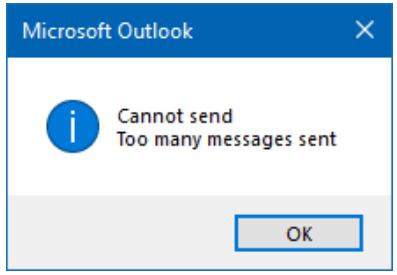
Can you do better than 4 bits for sending one message? What would it mean to do better?

How about transmitting n messages, each of which is one of 16 possible values? Does it matter that I transmit multiple messages?

Probability and Encoding

Suppose you need to send 1000 messages, each of which can be one of 16 possibilities. But *on average* 99.5% will be message 10. Does it still require $4 \times 1000 = 4000$ bits to send your 1000 messages? How could you do better?

It is possible to come up with an encoding that will do better on average than the naïve encoding. "On average" is important here, because it may do worse on some anomalous cases.



A Better Encoding

Msg	Encoding
0	10000
1	10001
2	10010
..	...
9	11001
10	0
11	11011
..	...
15	11111

Consider this encoding.

Given 1000 messages, on average 995 of them will be message 10, and 5 will be other messages.

Can you recover the original sequence of messages? How?

On average, this encoding takes $995 + (5 \cdot 5)$ bits or 1.02 bits per message. This compares to 4 bits per message for the naïve encoding.

Some Observations and Questions

Notice that we've gone from talking about the information content of a single *message* to talking about that of a *language* or *message stream*.

- Our encoding is pretty good, but can we do even better? Is there a limit to how well we can do?
- We often operate under incomplete information. Are the other 15 possibilities all equally likely?
- Computing the number of bits per message depends on knowing the *prior probabilities*—how often each message appears in an arbitrarily long sequence of messages.

Some Observations and Questions

- The "on average" part is important; some sequences would be less efficient under our encoding. How could that happen?
- We used the "naïve encoding" as our benchmark, but there are much worse encodings.

- Is it possible to find an *optimal* encoding? What would that mean?

What properties must a reasonable encoding have?

Encoding Properties

We insist that encoding be:

lossless: it must be possible to recover the entire sequence of messages from the transmission;

uniquely decodable: for any encoded string, there must be only one translation;

streaming: there must be no breaks in the encoding string.

Sufficient (but not necessary) for unique decodability is the property of being *prefix-free*: the string representing any symbol cannot be an initial prefix of the string representing any other symbol.

Unique Decodability

E.g., suppose you had a language containing the symbols A, B, C. The following is not a uniquely decodable encoding:

A	1
B	0
C	10

Explain why this is not uniquely decodable.

A good clue is that it does not have the prefix-free property.

Note: A language can fail to have the prefix-free property, but still be uniquely decodable. E.g., if the language above only had "A" and "C" it would be uniquely decodable. But parsing such a language may require arbitrary "look-ahead."

Exercise: Unique Decodability

Which of the following codes is uniquely decodable?

	A	B	C	D
code1	0	01	11	111
code2	0	01	110	111
code3	0	10	110	111
code4	1	10	110	111

Entropy

Assume you know the relative frequency of the various symbols in a language. Given the *best possible encoding* of the language, how many bits (on average) do you have to transmit to convey a messages of k symbols? What is the average number of bits per symbol sent? This number is called the *entropy* of the language.

Entropy H is computed as follows. If p_i is the probability of the ith symbol in the language, then

$$H = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i)$$

From now on, all log's are base 2.

Entropy Example 1

Consider the flipping of a *fair* coin. There are two equally likely outcomes, H and T, which we can represent as 0 and 1, respectively.

What is the entropy of this language?

I.e., how many bits are required on average to send one of a series of outcomes of the experiment of flipping the coin?

Would it be possible to do any better?

Entropy Example 1

Solution: There are two possible outcomes, each of probability 0.5. Then:

$$h = -(0.5 \times \log 0.5 + 0.5 \times \log 0.5) = 1$$

This says that on average we can't do any better than one bit to transmit the result of one toss of the coin. Think about why that is.

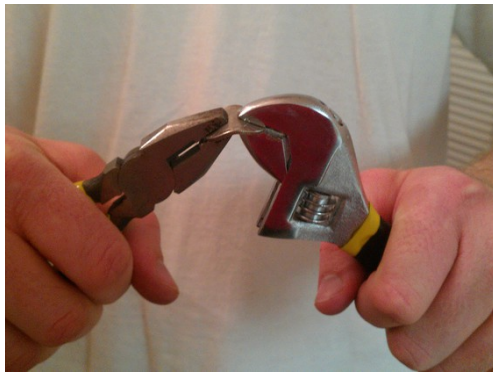
Therefore, the "naïve" encoding is actually the optimal encoding.

Entropy Example 2

Suppose we have an unbalanced coin that is three times more likely to yield a head than a tail.

What is the entropy of this language? Do you think it will be greater than 1 or less than 1? Why?

Remember what entropy means!



Entropy Example 2

Solution: There are two possible outcomes: H has probability 0.75 and T has probability 0.25. Then:

$$h = -(0.75 \times \log 0.75 + 0.25 \times \log 0.25) \approx 0.811$$

This says that it's theoretically impossible to encode this in a way that uses less than 0.811 bits (on average) to transmit the result of each toss of the coin. But is it possible to do better than 1 bit per toss?

Entropy Example 2

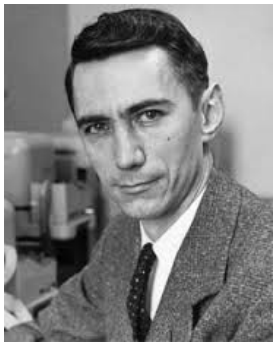
Solution: There are two possible outcomes: H has probability 0.75 and T has probability 0.25. Then:

$$h = -(0.75 \times \log 0.75 + 0.25 \times \log 0.25) \approx 0.811$$

This says that it's theoretically impossible to encode this in a way that uses less than 0.811 bits (on average) to transmit the result of each toss of the coin. But is it possible to do better than 1 bit per toss?

Note that this *doesn't give you an encoding* more efficient than the naïve encoding. Do we know that there is one? How can we possibly find it?

Fundamental Theorem



Claude Shannon proved a theorem, which we'll discuss shortly that says that: *there is always an encoding that arbitrarily closely approaches the entropy.*

But finding the encoding may require some ingenuity.

Entropy Example 2 Revisited

Notice: You can't do better than one bit per flip if you transmit the results in real time, i.e., as soon as each experiment completes. Instead, you have to collect and encode a *sequence* of results. Suppose we represent sequences of two flips at a time:

Result	Prob.	Code
HH	9/16	0
HT	3/16	10
TH	3/16	110
TT	1/16	111

How did we get those probabilities? Why that encoding?

Entropy Example 2 Revisited

Result	Prob.	Code
HH	9/16	0
HT	3/16	10
TH	3/16	110
TT	1/16	111

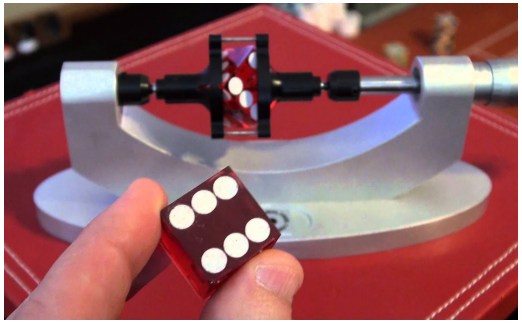
On average, we can represent a series of 32 flips with 27 bits, rather than 32 bits. Why? That yields an efficiency of 27/32 or 0.844. That's not a bad approximation of the entropy. (0.811)

Exercise: Improve this by coding sequences of three flips. What is the efficiency of that encoding? (.823) How about four flips?

Entropy Example 3

Suppose you have a six-sided die that is unbalanced such that:

- 1 and 2 are equally likely;
- 3 and 4 are equally likely;
- 5 and 6 are equally likely;
- 1 is twice as likely as 3;
- 3 is three times as likely as 5.



What are the relative probabilities of each outcome?

Entropy Example 3

1. What is the "naïve" encoding for this language?
2. What is the entropy of this language?
3. Find an encoding that is more efficient than the naïve encoding.
4. Argue that your encoding is more efficient than the naïve encoding.

An Aside: Randomness Extractors

Suppose you have a biased coin, like our coin that is three times more likely to yield a head than a tail. This sequence of flips is not *random*. But can you use this coin in such a way as to mimic a fair coin?

An Aside: Randomness Extractors

Suppose you have a biased coin, like our coin that is three times more likely to yield a head than a tail. This sequence of flips is not *random*. But can you use this coin in such a way as to mimic a fair coin?

Yes! Consider our unbalanced coin from Entropy Example 2:

Result	Prob.	Code
HH	9/16	0
HT	3/16	10
TH	3/16	110
TT	1/16	111

Notice that HT and TH are equally likely. Throw out all other outcomes and use these as 0 and 1.

An Aside: Randomness Extractors

John von Neumann gave such a construction. This is called a *randomness extractor* and sometimes allows you to derive a random sequence from non-random data.

The notion of randomness is very important and related to entropy. For example, if using a 128-bit key in a cryptographic algorithm, you want 128 bits of *entropy*, meaning that the key contains as much uncertainty as a 128-bit random string. This is seldom achieved.

Entropy Continued

As a special case, if we have n symbols, each of which is equally probable, then the probability of any of them is $1/n$. Then the entropy for the language is:

$$h = -(\log 1/n) = \log n$$

For example, a fair die with six sides has 6 equally likely outcomes. The entropy is:

$$h = -(\log 1/6) = \log 6 = 2.58$$

Hence, it requires 2.58 bits per symbol to transmit the result of a toss of the die.

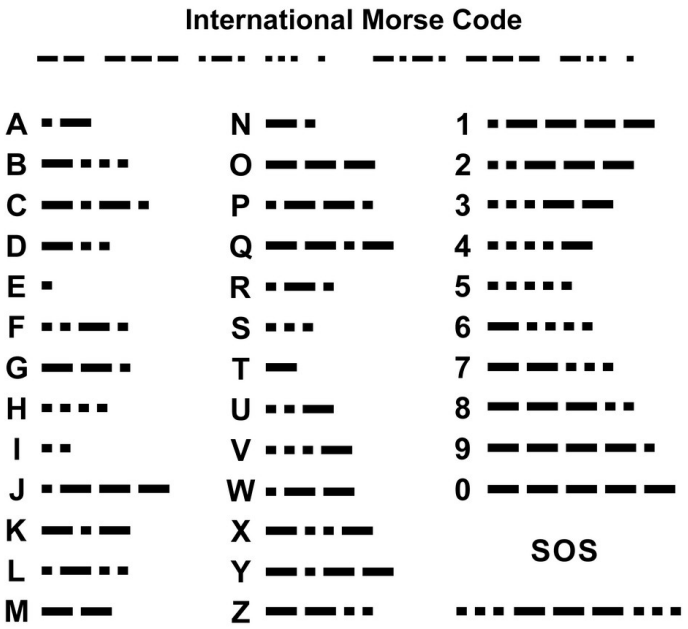
Finding a Coding

How do you come up with an efficient encoding? The idea is to use fewer bits for the items which occur most frequently, and more bits for those that occur less frequently. (Samuel Morse knew this instinctively. But is Morse code prefix-free? What gives?)

Huffman coding is guaranteed to find an efficient code for a given language *if* you know the probabilities of language units. In fact, it always uses *less than one bit per symbol more than the entropy,* which is extremely efficient.

The entropy gives the theoretical minimum number of bits to transmit the symbols in the language on average.

Morse Code



Entropy is Theoretical

Why is it that you can't usually transmit as efficiently as the entropy would predict? Consider sending one of 10 equally likely values. It is related to the different senses of "bit."

Entropy tells us what is possible in the long run. But...

"In the long run we are all dead." –John Maynard Keynes

Entropy Aside

Entropy measures the information content (or uncertainty in a transmission). For that reason, entropy is relative to a particular observer.

For example, before the winner of the Academy Award is announced, there are five possible winners. But for the auditors who stuffed the envelope, there isn't any uncertainty, hence no entropy. There may be more entropy for the Hollywood insider, and even more for the viewer at home.

Almost never are all outcomes equally likely, but the prior probabilities are often impossible to compute.

Discrete Events

The flipping of a coin or a die are *discrete* and *zero-memory* sources. A zero-memory source is one in which each symbol (event) is independent of the context (i.e., does not depend on the events that preceded it).

Information theory also addresses continuous behavior, but that's outside our scope.

Many language sources are far from being zero-memory. A source in which a symbol in a string depends on n preceding symbols is an *nth order Markov source*.

Efficient Encoding

Huffman coding is guaranteed to find an efficient code for a given language *if* you know the probabilities of language units. In fact, it always uses *less then one bit per symbol more than the entropy,* which is extremely efficient.

But, even if you know the entropy of a language, you may not know the characteristics of a given text. Eg., you may know the entropy of English, but not the characteristics of a given text file. You can use *adaptive coding* to deal with this.

Efficient Encoding: Lempel-Ziv

The Lempel-Ziv algorithm is used in many commercial text compression utilities such as `gzip`. It builds a dictionary on the fly according to the strings it encounters.

Lempel-Ziv is *asymptotically optimal*. That is, as the input text length tends to infinity, the compression approaches the optimal values predicted by information theory.

Encoding Larger Units

Often a better encoding is possible for a given language if the "units" of encoding are larger.

For example, suppose that you had 16 "letters" in the language, but there were only 1024 "words," each an average of 5 letters in length.

To encode on a letter-by-letter basis, you might use 4 bits per letter, and use an average $4 \times 5 = 20$ bits per word. But encoding the words directly requires only $(\log 1024) = 10$ bits per word.

Source Entropy

The entropy of a language is a measure of the most efficient possible encoding of the language.

The *entropy of a message source* is the amount of information content that the source can produce in a given period. This is measured in bits per second.

Any communication medium (channel) can transmit an arbitrary amount of information, given enough time and unbounded buffering capacity. But can a given channel transmit the information *in real time*?

Channel Capacity

The *capacity* of a channel is the number of bits that can be sent per second over the channel. This is a property of the communication medium *and may be reduced by noise on the channel*.

The *fundamental theorem of the noiseless channel* (Shannon) says the following: If a source has entropy h (bits per symbol) and a channel can transmit C bits per second, then it is possible to encode the signal is such a way as to transmit at an average rate of $(C/h) - \epsilon$ symbols per second, where ϵ can be made arbitrarily small. It is impossible to transmit at an average rate greater than C/h.

Noisy Channels

If a channel is noisy, it is possible to quantify the amount of uncertainty in the channel. The average uncertainty that x was transmitted when y was received is denoted $h_y(x)$ and is called the *equivocation* of the channel.

For a *symmetric* channel, the uncertainty does not depend on which symbol was sent. If a channel is not symmetric, the uncertainly varies depending on which symbol was sent.

The equivocation is the average uncertainty of the message recipient as to what message was actually sent.

Noisy Channels (Cont.)

A somewhat surprising theorem proved by Shannon is the following:

> *Let a discrete channel have a capacity C and a discrete source an entropy per second h. If $h < C$ there exists a coding system such that the output of the source can be transmitted over the channel with an arbitrarily small frequency of errors. If $h > C$ it is possible to encode the source so that the equivocation is less than $h - C + \epsilon$, where ϵ is arbitrarily small.*

It is crucial to understand that the channel capacity in this theorem is the *computed capacity as reduced by the noise*.

Noisy Channels (Cont.)

The upshot of this is that a message can be transmitted reliably over even a very noisy channel by increasing the redundancy of the coding scheme.

Here "efficiency" means the ratio of the number of data bits to the number of bits used to encode them.

Thus, covert channels in the system cannot be dismissed with the argument that they are noisy and hence useless. You can always get the message through by increasing redundancy.

Entropy vs. Redundancy

In an intuitive sense, the difference between the efficiency of the encoding and the entropy (theorical limit of efficiency) is a measure of the redundancy in the encoding. Another way of looking at this is: what is the most effective lossless compression algorithm imaginable?

If you can find an encoding with efficiency matching the entropy, *then there is no redundancy in the encoding*. We've seen at least one example of that. What was it?

The standard encoding for English contains a lot of redundancy. Hence the entropy is significantly lower than the encoding efficiency of the language.

Entropy vs. Redundancy

In an intuitive sense, the difference between the efficiency of the encoding and the entropy (theorical limit of efficiency) is a measure of the redundancy in the encoding. Another way of looking at this is: what is the most effective lossless compression algorithm imaginable?

If you can find an encoding with efficiency matching the entropy, *then there is no redundancy in the encoding*. We've seen at least one example of that. What was it?

The standard encoding for English contains a lot of redundancy. Hence the entropy is significantly lower than the encoding efficiency of the language.

Fr xmpl, y cn prbbly gss wht ths sntnc sys, vn wth ll f th vwls mssng. Tht ndcts tht th nfrmtn cntnt cn b xtrctd frm th rmnng smbls.

The Effect of Redundancy

Someone recently sent me this message. It illustrates the redundancy of English text:

> *Aoccdrnig to rscheearch at Cmabirgde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a ttoal mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Amzanig huh?*

Spammers count on the ability of humans to decipher such text, and the inability of computers to do so, when they scramble the words in a message header to defeat anti-spam software. *Care to order some Vi@gra or Vigara?*

Entropy of English: Zero-Order

Suppose we want to transmit English text (26 letters and a space). If we assume that *all characters are equally likely*, the entropy is:

$$h = -(\log 1/27) = 4.75$$

This is the *zero-order* model of English. (Notice that this also meets the "zero-memory" assumption mentioned before, though those are different notions.)

This gives an approximation to the entropy. If it gave the "real" entropy of English, then a message of n symbols could not be transmitted in less than $n \times 4.75$ bits. But the underlying assumption is clearly false.

Entropy of English: First-Order

In written or spoken English, some symbols occur much more frequently than others.

letter	frequency	letter	frequency	letter	frequency	letter	frequency
a	0.08167	b	0.01492	c	0.02782	d	0.04253
e	0.12702	f	0.02228	g	0.02015	h	0.06094
i	0.06966	j	0.00153	k	0.00772	l	0.04025
m	0.02406	n	0.06749	o	0.07507	p	0.01929
q	0.00095	r	0.05987	s	0.06327	t	0.09056
u	0.02758	v	0.00978	w	0.02360	x	0.00150
y	0.01974	z	0.00074				

Assuming that all symbols are *independent* of one another, but follow the probabilities above, the entropy is 4.219 bits per symbol. This is the "first-order" model of English (but still "zero-memory.")

Entropy of English: Higher-Order

The assumption of independence (zero memory) is also incorrect. For example, given a "Q" in English text, it is overwhelmingly likely that the next letter will be "U", much more likely than the probability of "U" in arbitrary English text. The following shows the most common digrams and trigrams in English.

Digrams	Trigrams
EN	ENT
RE	ION
ER	AND
NT	ING
TH	IVE
ON	TIO
IN	FOR
TR	OUR
AN	THI
OR	ONE

Entropy of English: Higher-Order

We can compute tables of the likelihood of *digrams* (two-letter combinations), *trigrams*, etc. Adding digrams to the computation gives a second-order model; adding trigrams gives a third-order model; etc.

A third-order model yields 2.77 bits per symbol. The actual entropy is the "limit" of this process of taking higher and higher order models.

Estimates by Shannon based on human experiments have yielded values as low as 0.6 to 1.3 bits per symbol.

What's Wrong With This?

Recall that: Lempel-Ziv is *asymptotically optimal*. That is, as the input text length tends to infinity, the compression approaches the optimal values predicted by information theory.

Suppose I wanted to compute the "actual" entropy of English. Couldn't I do the following:

- Get a very large volume of English text.
- Run it through the Lempel-Ziv algorithm.
- Compute the bits/symbol of the result.

What's wrong with this argument?

Entropy of Arbitrary Languages

Any computed estimate of the entropy of English is based on averages in large volumes of text. But averages may not apply to specific texts.

Consider the following text from the 267-page novel *Gadsby*, by Earnest Vincent Wright (1939).

> *Upon this basis I am going to show you how a bunch of bright young folks did find a champion; a man with boys and girls of his own; a man of so dominating and happy individuality that Youth is drawn to him as is a fly to a sugar bowl. It is a story about a small town. It is not a gossipy yarn; nor is it a dry, monotonous account, full of such customary "fill-ins" as "romantic moonlight casting murky shadows down a long winding country road."*

Entropy and Randomness

Note that entropy can be used to measure the amount of "redundancy" in the encoding. If the information content of a message is equal to the length of the encoded message, there is no redundancy.

Some sources define a *random* string as one that cannot be represented any more efficiently. I.e., no compression is possible for a truly random string.

Redundancy and Cryptography

Redundancy is crucial in understanding written or spoken communication. It also plays a vital role in cryptography.

Any knowledge you can discover about the plaintext of an encoded message provides help in decoding it. Why?

Redundancy and Cryptography

Redundancy is crucial in understanding written or spoken communication. It also plays a vital role in cryptography.

Any knowledge you can discover about the plaintext of an encoded message provides help in decoding it. Why?

1. The less redundancy in the plaintext, the harder the cryptanalyst's task.
2. For that reason, it is useful to compress *before* encrypting.
3. A good encryption algorithm will ensure that redundancy in the plaintext is not mirrored in the ciphertext.
4. In fact, a really good algorithm yields output that is indistinguishable from random, which means that it is useless to compress *after* encrypting.