CS 811 Final Project

Ray Shulang Lei
200253624
Department of Computer Science
University of Regina
Regina, Saskatchewan, S4S0A2, Canada

April 25, 2011

Abstract

This project paper reviews Claude E. Shannon's information theory and the algorithmic information theory.

1 Introduction

2 Claude E. Shannon's information theory

2.1 The entropy of information

In information theory, the entropy of information means the measurement of the uncertainty of the information. The entropy H of an variable X is:

$$H(X) = E(I(X))$$

Where I is the information content of X, and E is the expected value.
Let $x_i \in X$, and $p(x_i)$ be the probability mass function of X, then the information content of X is:

$$I(X) = -log_2 p(x_i)$$

Then the entropy can be written as[1]:

$$H(X) = \sum_{i=1}^{n} p(x_i)I(x_i) = -\sum_{i=1}^{n} p(x_i)log_2 p(x_i)$$

Note that each $p(x_i)I(x_i)$ gives the expect value, where $I(x_i)$ is the average entropy of the specific event outcome. Here we assume the unit of entropy is bit, therefore the logarithm base is 2.

The information gain of an event is simply the entropy difference before and after the event:

$$H_{gained}(X) = H_{before}(X) - H_{after}(X)$$

2.2 The fair coin tossing example

When tossing a coin, the outcomes are either head(x_h) or tail(x_t). Thus

$$X = \{x_h, x_t\}$$

Thus, before tossing the coin once, its entropy is:

$$H_{before}(X) = -(p(x_h)log_2 p(x_h) + p(x_t)log_2 p(x_t))$$

Assume this is a fair coin:

$$p(X = x_h) = 0.5, p(X = x_t) = 0.5$$

Therefore,

$$H_{before}(X) = -(0.5 \times log_2 0.5 + 0.5 \times log_2 0.5) = -(-0.5 - 0.5) = 1$$

Which means before toss the coin, the event would have 1 bit entropy.
After the coin has been tossed, let's say the result is tail, then its possible outcome now is only x_t:

$$X_t = \{x_t\}, p(X_t = x_t) = 1$$

And its entropy is now:

$$H_{after}(X_t) = -(1 \times log_2 1 + 1 \times log_2 1) = -(0 + 0) = 0$$

Thus, the entropy we have gained during the coin tossing is:

$$H_{gained}(X) = H_{before} - H_{after} = 1 - 0 = 1$$

Which is 1 bit entropy. This means we need 1 bit entropy to store the each toss of a fair coin.
But what if the coin we toss is not fair?

2.3 The unfair coin tossing example

Assume we have an unfair coin that the chance it lands on its head is 0.8 and the chance it lands on its tail is 0.2:

$$p(X = x_h) = 0.8, p(X = x_t) = 0.2$$

Thus, before tossing the coin once, its entropy is:

$$H_{before}(X) = -(p(x_h)log_2p(x_h) + p(x_t)log_2p(x_t))$$

Which equals to:

$$H_{before}(X) = -(0.8 \times log_20.8 + 0.2 \times log_20.2)$$

$$\approx -(0.8 \times -0.321928094887362 + 0.2 \times -2.321928094887362)$$

$$= -(-0.25754247590989 - 0.464385618977472)$$

$$= 0.721928094887362$$

Again, the entropy after the coin has been tossed with a result of head is 0:

$$X_h = \{x_h\}, p(X_h = x_h) = 1$$

And its entropy is now:

$$H_{after}(X_h) = -(1 \times log_21 + 1 \times log_21) = -(0 + 0) = 0$$

Thus, the entropy we have gained during the unfair coin tossing is:

$$H_{gained}(X) = H_{before} - H_{after} = 0.721928094887362 - 0 = 0.721928094887362$$

We can see that when the possibilities are not even, the entropy is 0.721928094887362 bit, which is less than 1 bit when the coin is fair.

2.4 The average information gain with specific outcome

The above two examples gives us how many information on average we would gain during a fair coin toss event and a unfair coin toss event.
Notice that these information gains are average information gains for each coin tosses considering both outcomes of heads and tails.

What is the average information gain for the coin toss event that the outcomes are heads only or tails only, then? Let's look at the following examples: We will look into I(X) where I is the information content of X.

$$I(X) = -log_2 p(x_i)$$

In the case of a fair coin toss with outcomes of heads only:

$$I(X) = -log_2 p(x_h), p(x_h) = 0.5$$

$$\implies I(X) = -log_2 \frac{1}{2} = 1$$

In the case of a fair coin toss with outcomes of tails only:

$$I(X) = -log_2 p(x_t), p(x_t) = 0.5$$

$$\implies I(X) = -log_2 \frac{1}{2} = 1$$

Thus 1 bit entropy is gained on average with each fair coin toss with the outcome of both head and tail.

In the case of the unfair coin toss with outcomes of heads only:

$$I(X) = -log_2 p(x_h), p(x_h) = 0.8$$

$$\implies I(X) = -log_2 \frac{4}{5} \approx 0.321928094887362$$

In the case of the unfair coin toss with outcomes of tails only:

$$I(X) = -log_2 p(x_t), p(x_t) = 0.2$$

$$\implies I(X) = -log_2\frac{1}{5} \approx 2.321928094887362$$

When the outcome is head, we only gained 0.3 bit of information, where when the outcome is tail, we gained 2.3 bits of information. We discovered that more entropy is gained on average with each unfair coin toss event with the outcome of less common. We conclude that when the outcome of an event is less common, then it contains more information.

2.5 Exercise 1 - Rolling a fair die

What is the average entropy gain for the event of rolling a fair die? (in binary bits)

2.6 The Coding theory

By measuring entropy, it's possible that we can compress data and exploit better coding. After finished exercise 1, we can see that the average entropy gain for the event of rolling a fair die is about 2.58 bits. How many bits of storage we need to save this information? Let's try using two bits first:

$$00 - x_1, 01 - x_2, 10 - x_3, 11 - x_4, ?? - x_5, ?? - x_6$$

Apparently, with two bits of storage, we can only save 4 out of 6 outcomes. Let's increase to three bits:

$$000{-}x_1, 001{-}x_2, 010{-}x_3, 011{-}x_4, 100{-}x_5, 101{-}x_6, 110{-}unused, 111{-}unused$$

Now we can save all the outcomes. However, 110 and 111 are unused and wasted. This is becase with three bits storage, we have:

$$H(X) = -\sum_{1}^{8}\frac{1}{8}log_2\frac{1}{8} = 3$$

Which is capable of saving 3 bits of entropy. For the event of rolling a fair die, only 2.58 bits of entropy are needed.
Let's look at an example of rolling a fair die 8 times with the outcomes of

$x_2, x_4, x_5, x_3, x_6, x_4, x_3, x_1$ in sequence. Using the three bits code mapping above we can write this as:

$$001011100010101011010000$$

Now let's look at another coding for the 6 outcomes:

$$010 - x_1, 011 - x_2, 10 - x_3, 11 - x_4 000 - x_5, 001 - x_6,$$

And let's encode our example of rolling a fair die 8 times with the outcomes of $x_2, x_4, x_5, x_3, x_6, x_4, x_3, x_1$ in sequence:

$$01111000100011110010$$

We store 8 outcomes with 20 bits, compare to:

$$001011100010101011010000$$

where we only used 24 bits. This is possible because of the new code mapping:

$$010 - x_1, 011 - x_2, 10 - x_3, 11 - x_4 000 - x_5, 001 - x_6,$$

$$\begin{aligned} H(X) &= \frac{1}{6} \times 3 + \frac{1}{6} \times 3 + \frac{1}{6} \times 2 + \frac{1}{6} \times 2 + \frac{1}{6} \times 3 + \frac{1}{6} \times 3 \\ &= \frac{1}{2} + \frac{1}{2} + \frac{1}{3} + \frac{1}{3} + \frac{1}{2} + \frac{1}{2} \approx 2.67 \end{aligned}$$

Thus, we use about 2.67 bits for each outcome on average, instead of 3 bits. For example, in English, words being used more frequently are coded with less letters than words being used rarely. Such as {'I', 'is', 'a'} compare to {'communism', 'encephalopathy'}

2.7 Exercise 2 - entropy of languages

Try to explain why languages like English is more effecient to be inputed by a keyboard than languages like Chinese using Shannon's information theory.

Assumptions:
A typical keyboard device has about 50 keys related to language inputs.
Each English letter is being used evenly, and here we only considering 26 English alphabets plus peroid and comma.
There are about 3000 commonly used Chinese characters and they are being used evenly.

3 Algorithmic information theory

We can see that Shannon's information theory look at information in a probablistic and statistic way. By doing exercise 2, we can also see that using entropy to encode a language is tedious and troublesome. Even calculating the entropy of a language is hard, considering the probabilitic usage of different letters or characters are not the same. In reality, English has an entropy of 4.03 bits per word and Chinese has 9.65 bits per character. Calculating these entropies requires tremendous effort, not to mention establishing a coding schema.

Now let's look at a more descriptive and logical approach for the definition of information using Turing Machine's power.[2]

3.1 Minimal length descriptions

Suppose we have two piece of information represented as binary strings:

$$A = 01010101010101010101010101010101010101$$

$$B = 11100101101000111010100011101001101$$

Simply applying Shannon's information theory, by the size of the storage space, we might say that A has more information because its storage space is sufficient for more entropy than B. However, using intuition, we find that B actually contain more information than A:

A: repeat '01' 20 times

B: The information is '11100101101000111010100011101001101'

If we apply coding theory to these strings we would find out that indeed B has more information than A. However, using descriptive way to represent information is more logical for human to understand than finding some coding schema to match the inside pattern of some information. Since we have a definition of algorithm as a turing machine now, we can descript any

information using turing machines. And we can view the optimalized representation of a piece of information as the minimal length description of this piece of information as a turing machine:

$$< M, w >= 11001111001100...1100 \mid 01 \mid 01101011...010$$

11001111001100...1100 is the description of turing machine$< M >$ where every bit is doubled, 01 is the delimiter, and 01101011...010 is the input string w.

3.2 Descriptive complexity

The descriptive complexity of x is the length of the minimal description of x:

$$K(x) = |d(x)|$$

Where d(x) is the minimal description of x, K(x) is the descriptive complexity of x.

The descriptive complexity of x is less than or equal to the length of x plus a constant number c, which is universal and not dependent on the string x:

$$K(x) \leq |x| + c \qquad \alpha$$

Proof. We can construct a turing machine M, which it only outputs its input. Since the length of this turing machine is fix and insignificantly compare to the length of its input, we can say that the upper bound of the Descriptive complexity of a piece of information is insignificantly more then the length of itself. □

The upper bound of the descriptive complexity of xx is insignificantly more than the descriptive complexity of x:

$$K(xx) \leq K(x) + c \qquad \beta$$

Proof. To represent a repetitive information, we can construct a Turing machine which says: Repeat the information of "..." (N) more time(s). □

The upper bound of the descriptive complexity of combining two piece information together is:

$$K(xy) \leq 2K(x) + K(y) + c \qquad \gamma$$

Proof. we can construct A Turing machine M on its input w. M breaks input w into two parts, the description of x, denoted as d(x), with all bits doubled, a delimiter '01', and d(y).

□

For any description language p, a fixed constant c exists that depends only on p, for all string x:

$$K(x) \leq K_p(x) + c$$

Where $K_p(x)$ is a description of x in language p.

Proof. Denote description of x in LISP as $d_{lisp}(x)$.
A TM M on input $d_{lisp}(x)$:
1. M interpret LISP;
2. M outputs x.
$c = | < M > |$

□

3.3 Compressible and incompressible

First let's look at these definitions:

Definition 1. *If $K(x) \leq |x| - c$, then x is c-compressible;*

Definition 2. *If $K(x) > |x| - c$, then x is incompressible by c;*

Definition 3. *If $K(x) \geq |x|$, then x is incompressible.*

Now let's look at some facts about compressible and incompressible:

Statement 1. *Incompressible strings of every length exist.*

Proof. For each length n, there are 2n possible strings, and there are 1+2+4+8+...+2n-1 = 2n-1 possible descriptions. Therefore at least one string of length n is incompressible. □

Statement 2. *At least* $2^n - 2^{n-c+1} + 1$ *strings of length n are incompressible by c.*

Proof. For each length n, at most $2^{n-c+1} - 1$ strings are c-compressible. The remaining $2^n - (2^{n-c+1} + 1)$ are incompressible by c. □

To understand Statement 3, we first need to look at following two definitions:
Definition 1. *A property of x, is a function f, such that f(x) only returns true or false.*
Definition 2. *If we say A property holds for almost all strings, then as the length n grows, the fraction of the strings set A{(a)}, in which f(a) = false, approaches 0.*

Now let's look at this statement:
Statement 3. *There exists a computable property f that holds for almost all strings. Then, for all b ¿ 0, f returns false on limited number of strings that are incompressible by b.*

Proof. A TM M on input i, where i is a binary integer:
1. Find the ith string s where f(s) = false. All strings are generated with lexical orders.
2. Output string s.

Then, for any string x where f(x) = false, $< M, i_x >$ is description.

Get any number $b > 0$. Select n such that $1/2^{b+c+1}$ fraction of strings of length n or less fail to have property f. Let x be a string in length n that fails f. We have $2^{n+2} - 1$strings of length n or less. Thus:
$i_x \leq (2^{n+1} - 1)/(2^{b+c+1}) \leq 2^{n-b-c}$

Therefore $|i_x| \leq n - b - c$, so $| < M, i_x > |$ is at most $(n - b - c) + c = n - b$, $K(x) \leq n - b$

Which exactly means every x fails f is compressible by b.

□

How do we find incompressible strings?

Statement 4. *For some constant numbers b, for every string x, the minimal description d(x) of x is incompressible by b.*

Proof. A TM M, on input $< R, y >$, R is a TM and y is string:
1. Run R on y. If output is not the form of $< S, z >$, reject;
2. Run S on z, halt with output on tape.

Let $b = | < M > | + 1$. Assume that d(x) is b-compressible, then:
$|d(d(x))| <= |d(x)| - b$
The length of $< M, d(d(x)) >$ is at most :
$| < M > | + |d(d(x))| <= (b - 1) + (|d(x)| - b)$
$| < M > | + |d(d(x))| <= |d(x)| - 1$
But, $< M, d(d(x)) >$ is a description of x, denoted as d(x):
$|d(x)| <= |d(x)| - 1$
This is impossible! Thus this is contradiction, and d(x) can not be b-compressible for all strings x.

□

4 Exercise Solutions

4.1 Exercise 1 Solution

First, we list the domain of the outcomes of rolling a fair die:

$$X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$$

Second, we list the possibility distribution:

$$p(X = x_1) = \frac{1}{6}$$

$$p(X = x_2) = \frac{1}{6}$$
$$p(X = x_3) = \frac{1}{6}$$
$$p(X = x_4) = \frac{1}{6}$$
$$p(X = x_5) = \frac{1}{6}$$
$$p(X = x_6) = \frac{1}{6}$$

And then we calculate the entropy before we roll the die:

$$H_{before}(X) =$$
$$-p(x_1)log_2p(x_1) - p(x_2)log_2p(x_2)$$
$$-p(x_3)log_2p(x_3) - p(x_4)log_2p(x_4)$$
$$-p(x_5)log_2p(x_5) - p(x_6)log_2p(x_6)$$

$\Longrightarrow$

$$H_{before}(X) =$$
$$-\frac{1}{6}log_2\frac{1}{6} - \frac{1}{6}log_2\frac{1}{6}$$
$$-\frac{1}{6}log_2\frac{1}{6} - \frac{1}{6}log_2\frac{1}{6}$$
$$-\frac{1}{6}log_2\frac{1}{6} - \frac{1}{6}log_2\frac{1}{6}$$

$\Longrightarrow$

$$H_{before}(X) = -log_2\frac{1}{6}$$

$\Longrightarrow$

$$H_{before}(X) = 2.584962500721156$$

We also need to calculate the entropy after we roll the die. Let's say the outcome is x_i:

$$X_i = \{x_i\}$$

Then we figure the possibility distribution:

$$p(x_i) = 1$$

Then the entropy:

$$H_{after}(X_i) = -p(x_i)log_2p(x_i)$$
$$\implies H_{after}(X_i) = -1log_21 = 0$$

Thus the entropy gain of rolling a die is:

$$H_{gained}(X) = H_{before}(X) - H_{after}(X_i) = 2.584962500721156 - 0 = 2.584962500721156$$

We say that the entropy gain for the event of rolling a fair die is about 2.58 bits.

4.2 Exercise 2 Solution

A typical keyboard device has about 50 keys related to language inputs. Thus it's average entropy for each key stroke is:

$$H(X_k) = -\sum_{i=1}^{50} p(x_i)log_2p(x_i)$$

$$\implies H(X_k) = -\sum_{i=1}^{50} \frac{1}{50}log_2\frac{1}{50}$$
$$= log_250 \approx 5.643856189774724$$

Which is about 5.6 bits entropy for each key stroke.
Now let's calculate the average entropy for each letter of English. Assuming their possibility distribution is even and considering 26 English alphabets plus peroid and comma:

$$H(X_e) = -\sum_{i=1}^{28} \frac{1}{28}log_2\frac{1}{28}$$

$$= log_228 \approx 4.807354922057604$$

Thus we have about 4.8 bits entropy for each letter in English.
Storing each letter in English with 5.6 bits of entropy per key stroke is sufficient. Thus we can have one-to-one mapping between keys and letters and users do not need to remember any coding. We can see that English and

keyboard are naturally fit.

Let's look at entropy of each character of Chinese. Assuming there are about 3000 commonly used Chinese characters and their possibility distribution is even:

$$H(X_c) = -\sum_{i=1}^{3000} \frac{1}{3000} log_2 \frac{1}{3000}$$
$$= log_2 3000 \approx 11.55$$

Thus we have about 11.55 bits entropy for each character in Chinese.
Storing each letter in Chinese with 5.6 bits of entropy per key stroke is insufficient. Thus we need a coding schema containning more than three key strokes for each Chinese character, and users have to remember the coding schema in order to use the keyboard as an input device. In fact, Chinese Pinyin is being use as a such schema, where Chinese Pinyin is very close to English compare to traditional Chinese.

References

[1] Fazlollah M. Reza, *An Introduction to Information Theory*. ISBN 0-486-68210-2, Dover Publications, Inc., New York, Dover Edition, 1994.

[2] Micheal Sipser, *Introduction to The theory of Computation*. ISBN 0-534-95097-3, THOIVISON COURSE TECHNOLOGY, Boston, Massachusetts, 02210. Second Edition, 2006.